

**Universidad Tecnológica de Pereira  
Ingeniería de Sistemas y Computación  
Computación de Alto Desempeño  
Filtro de Sobel con CUDA y OpenCV  
Segundo Parcial**

Angie Vanessa Penagos Rios  
*Ingeniería de Sistemas y Computación*  
*Universidad Tecnológica de Pereira*  
*Email: angie-19-v@utp.edu.co*

Diego Alejandro Valencia Bermúdez  
*Ingeniería de Sistemas y Computación*  
*Universidad Tecnológica de Pereira*  
*Email: valencia\_95\_@utp.edu.co*

Daniela Zuluaga Ocampo  
*Ingeniería de Sistemas y Computación*  
*Universidad Tecnológica de Pereira*  
*Email: daniela-zuluaga@utp.edu.co*

**Resumen**—El procesamiento de imágenes tiene como objetivo mejorar o resaltar aspectos dentro de una imagen para elaborar diferentes estudios; la binarización, la manipulación de contraste y el filtrado de imágenes son algunos de los algoritmos de procesamiento de imágenes más comunes. En este documento se tiene como objetivo realizar una comparación en tiempos de los diferentes modelos de memoria utilizando CUDA para realizar el algoritmo de filtro de Sobel, también se realizará una comparación de todos los modelos de memoria contra el algoritmo de filtro de Sobel ofrecido por OpenCV.

## 1. Introducción

Desde 1972, con el lanzamiento del satélite Landsat el procesamiento de imágenes se convirtió en un aporte importante para diferentes áreas del conocimiento tales como la medicina y la milicia. Tiempo después con la aparición de las computadoras de alta capacidad y memoria, este campo comenzó a crecer hacia diferentes aplicaciones como análisis de terrenos, aplicaciones en estudio de oleaje, reconocimiento de materiales en arqueología, entre otras. Por otro lado la constante demanda de mayor poder computacional para responder a las necesidades de la comunidad científica como lo es el procesamiento de imágenes para el tratamiento de datos ha fomentado a realizar avances en el software y hardware, creando así la computación masivamente paralela para responder a las necesidades de procesamiento que se pueden presentar.

En la actualidad la computación masivamente paralela tiene como objetivo mejorar el tiempo de ejecución de procesos ya existente utilizando el hardware y el software necesario. En este documento se estudia el uso de la computación masivamente paralela para la optimización de uno de los algoritmos de procesamiento de imágenes más comunes como lo es el filtro de Sobel, este algoritmo busca detectar los bordes de una imagen y resaltar los mismos

para segmentar la imagen, reconocer los objetos existentes en la imagen, entre otras utilidades. Para realizar este estudio se usa la arquitectura de cálculo paralelo de nVidia llamado CUDA, este nos ofrece la posibilidad de realizar el algoritmo del filtro de sobel paralelizado con 3 modelos de memoria diferentes como lo son la memoria compartida, la memoria constante y la memoria global.

## 2. Métodos

### 2.1. Escala de grises

Una imagen es una matriz de puntos donde en cada punto hay un color representado por bits, estos bits contienen una escala para cada color primario, Rojo, Verde y Azul(RGB) representados por 8 bits cada uno, por ejemplo, para obtener grises el procedimiento que se realiza es poner cada color primario en la misma escala, multiplicando cada bit de la imagen RGB los valores de 0.299, 0.587 y 0.144 respectivamente. Este proceso de cambio de escala de grises se realiza con la finalidad de obtener una imagen suavizada que permita un mejor manejo para futuros procesamiento de la misma, como por ejemplo realizar un filtro como el de Sobel para obtener los bordes de la imagen. En la figura (1) se puede observar los resultados de este algoritmo.

Figura 1. Proceso de escala de grises



### 2.2. Convolución

Una convolución se define como un operador matemático que realiza una transformación de dos funciones diferen-

tes en una función que representa la superposición de las dos funciones anteriores, la función resultante es conmutativa, asociativa, y distributiva respecto a la suma. Este operador es utilizado dentro del algoritmo de filtro de Sobel para realizar la transformación de la imagen y resaltar cada uno de los bordes de la misma.

### 2.3. Filtro de Sobel

Un borde dentro de una imagen es una transición entre dos regiones diferentes de esta, el algoritmo del filtro de Sobel tiene como objetivo realizar la detección de estos bordes por medio de una mascara de convolución, esta mascara se calcula por medio de el calculo del gradiente de la imagen en dos direcciones (X y Y). El poder detectar los bordes de la imagen es de gran importancia ya que puede ofrecer mucha información acerca de una imagen como puede ser los contornos de los objetos para realizar una clasificación de las mismas, el reconocimiento de fronteras dentro de la imagen, entre otras.

Para la implementación de este algoritmo se realizó el suavizado de la escala de grises y a continuación se aplicó una convolución de la imagen convertida a escala de grises utilizando las mascaras de convolución definidas a continuación:

Cuadro 1. MASCARAS DE CONVOLUCIÓN

$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$	$\begin{matrix} -1 & -2 & -1 \\ 0 & 1 & 0 \\ 1 & 2 & 1 \end{matrix}$
Fila	
Columna	

En la figura (2) se puede observar los resultados de este algoritmo.

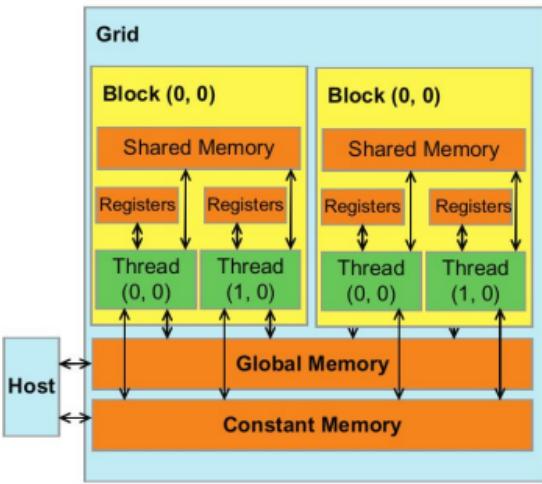
Figura 2. Filtro de sobel



### 2.4. Modelo de memorias de CUDA

Dentro de una GPU existen dos tipos de memorias las cuales son usada para el procesamiento de las operaciones, estas son la memoria *on-chip* y la memoria *off-chip*. La memoria *on-chip* se encuentra dentro de la GPU, por lo tanto es posible realizar un acceso más rápido, mientras que la memoria *off-chip* se encuentra fuera de la GPU lo que hace que acceder a esta se convierta en una tarea mas lenta. Debido a estos tipos de memoria CUDA establece 3 modelos de memoria, en la figura (3) se observa su distribución dentro de la GPU.

Figura 3. Memorias



**2.4.1. Memoria Global.** Es una memoria de lectura/escritura, esta se encuentra en la clasificación de *off-chip* lo que la convierte en una memoria de acceso lento, una característica que posee esta memoria es que todos los hilos que se encuentren en ejecución pueden acceder a esta y su tiempo de vida se da durante toda la ejecución del programa, por lo tanto es posible que las variables que se encuentren en esta memoria puedan ser usadas en diferentes espacios de procesamiento para transmitir la información.

**2.4.2. Memoria Compartida.** Es una memoria de lectura/escritura que se encuentra en la clasificación de *on-chip* por lo que el acceso a esta es más rápido, su tiempo de vida se da durante la ejecución de la función en donde se encuentra declarada y tiene como característica que es una memoria común a todos los hilos que se encuentren en un bloque, lo que permite una comunicación entre los mismos.

**2.4.3. Memoria Constante.** Es una memoria de lectura que se encuentra en la clasificación de *off-chip*, aunque posee una memoria cache en *on-chip* lo que la convierte en una memoria de acceso rápido; el ciclo de vida de esta memoria es durante toda la ejecución del programa y puede ser leída por todos los hilos que se estén ejecutando.

Para este proyecto se realizaron 4 implementaciones en donde se utiliza todos métodos descritos anteriormente, a partir de estas implementaciones se realizaron pruebas con 10 imágenes de diferentes tamaños, tomando 20 veces cada uno de los tiempos de ejecución de las implementaciones.

## 3. Resultados

Los resultados promedio de la medición de tiempos se encuentran a continuación en el cuadro 1.

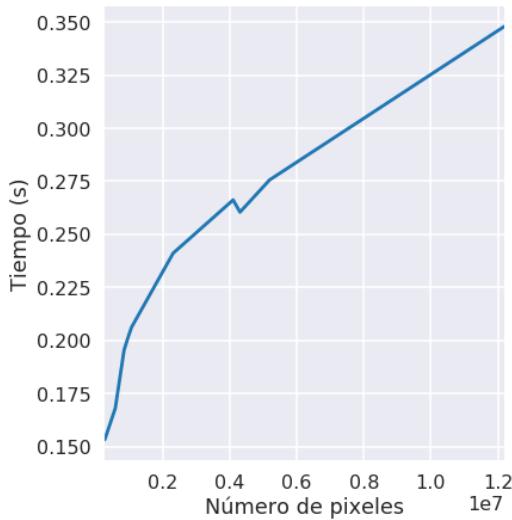
Cuadro 2. TIEMPOS PROMEDIOS DE EJECUCIÓN

Píxeles	Tiempos(segundos)			
	Secuencial	Global	Constante	Compartida
262144	0.15324205	0.00035565	0.00017215	0.00018005
576000	0.16800455	0.0007606	0.00034725	0.00036505
838810	0.19547565	0.0010936	0.00047335	0.00052445
972288	0.20211135	0.0013527	0.000582	0.00064645
1058400	0.2060778	0.00138905	0.000588	0.00065215
2304000	0.24107535	0.0027149	0.00107365	0.0012719
4096000	0.2661607	0.0047276	0.0018864	0.0021442
4300220	0.2604202	0.00507365	0.00199655	0.0022595
5184000	0.2755889	0.0059655	0.00233685	0.0026669
12192768	0.3478871	0.01322555	0.00533615	0.00611525

Los tiempos obtenidos con el algoritmo secuencial (de la librería OpenCV) tuvieron tiempos promedio que van desde 0.15s hasta 0.34s dependiendo del tamaño de la imagen, se resalta que el algoritmo secuencial es el que requirió más tiempo para cambiar la imagen a escalas de grises y realizar el filtro de Sobel, esto debido a que no utiliza al máximo las ventajas que brinda la GPU.

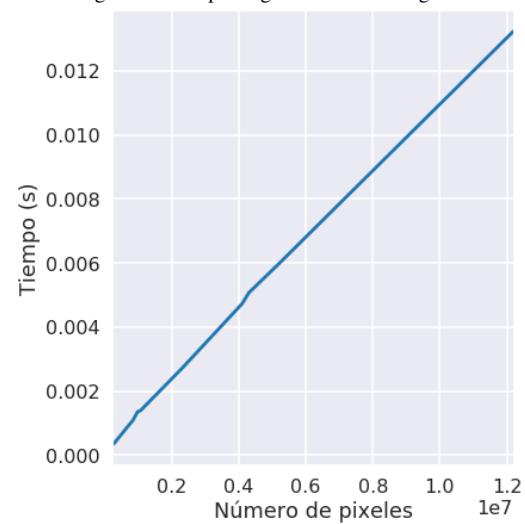
Para visualizar el comportamiento del algoritmo secuencial con las diferentes imágenes y tamaños de estas, se construyó la figura (4) con los promedios de los tiempos tomados para cada imagen, en esta se observa un crecimiento en el tiempo cuando el tamaño de las imágenes aumenta, pero en comparación a los demás algoritmos es el mas irregular, es decir, para algunas imágenes resultaron tiempo menores que en otras que era de mayor tamaño.

Figura 4. Tiempos algoritmo secuencial



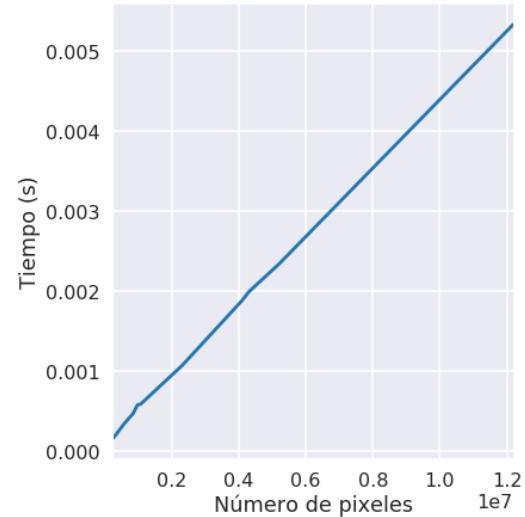
En la memoria global los tiempos fueron desde 0.00035s hasta 0.013s, igualmente dependiendo del tamaño de la imagen, en la figura (5) se observa un crecimiento constante en el tiempo con relación al tamaño de la imagen y se obtuvieron tiempos menores que en la implementación secuencial debido a que se aprovecha la programación masivamente paralela para la realización del filtro y el cambio de color a escalas de grises en la imagen.

Figura 5. Tiempos algoritmo memoria global



Con la memoria constante se obtuvieron mejores resultados en tiempo de procesamiento de la imagen que en los otros algoritmos, se obtuvieron tiempos que van desde 0.00017s a 0.0053s dependiendo del tamaño de la imagen, al igual que se observó un crecimiento muy parecido en el tiempo al que se obtuvo en el algoritmo de memoria global como se evidencia en la figura (6).

Figura 6. Tiempos algoritmo memoria constante



En la figura (7), se puede ver que el crecimiento de esta en el tiempo con relación al tamaño de las imágenes es el mas constante de todos los algoritmos, los tiempos obtenidos van desde 0.00018s a 0.006s, resultados muy similares a los de memoria constante, en esta implementación se hace uso de la memoria compartida que nos brinda una gran ventaja al poder compartir memoria con diferentes hilos del bloque.

Son muy buenos los resultados en el tiempo promedio y estos pueden ser aún mejores con una optimización del código.

Figura 7. Tiempos algoritmo memoria compartida

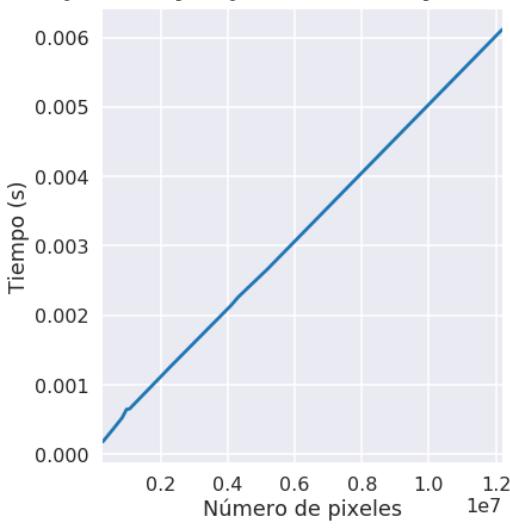
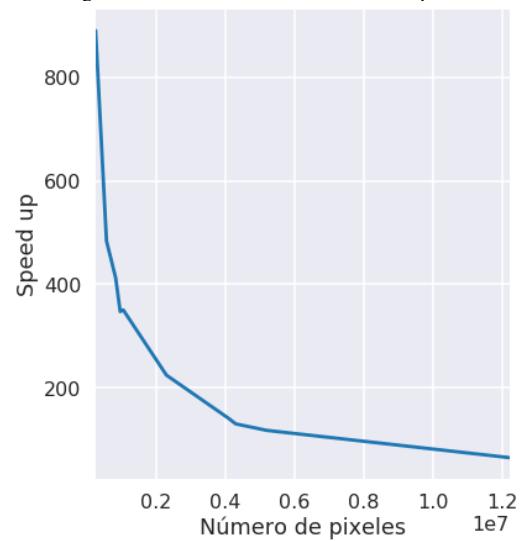


Figura 9. Aceleración con memoria compartida



Se realizaron gráficas de aceleración para los tres tipos de memorias, las gráficas resultantes se encuentran a continuación:

Se puede observar que la ganancia obtenida con el uso de memorias fue mayor en las memorias compartidas y constante y que esto tiene mayor relevancia cuando el tamaño de las imágenes crece, donde se nota que el tiempo de procesamiento requerido para imágenes pequeñas es mayor que el de imágenes con mayor número de píxeles.

Figura 8. Aceleración con memoria compartida

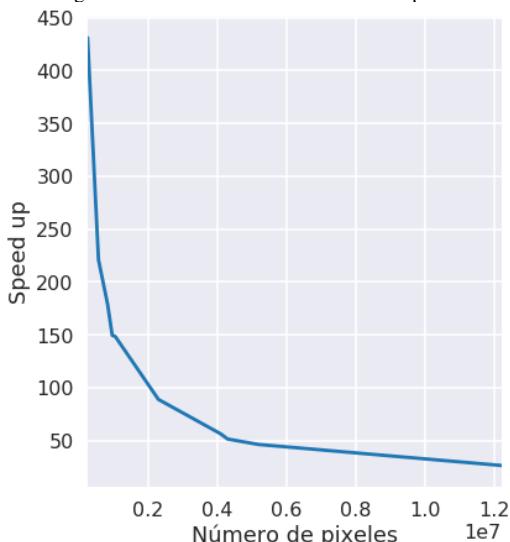
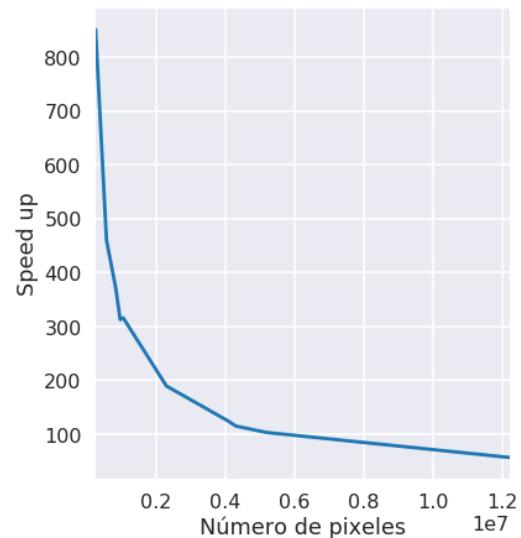
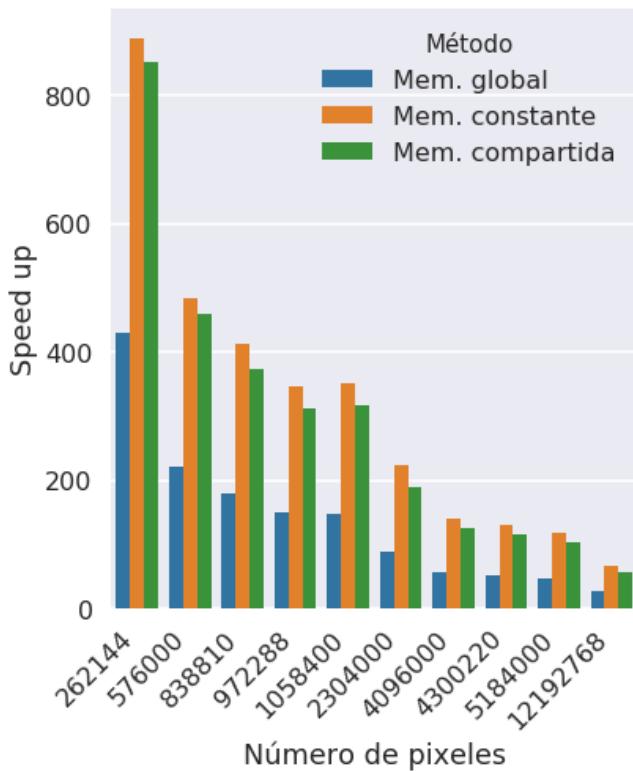


Figura 10. Aceleración con memoria compartida



A continuación se presenta en la figura (11), un gráfico de barras donde se compara las diferentes aceleraciones obtenidas para cada algoritmo, en este se evidencia mejor desempeño en las implementaciones de memoria constante y compartida frente a la memoria global, al igual que se nota una ganancia cuando las imágenes incrementan su tamaño.

Figura 11. Comparación entre aceleraciones



#### 4. Conclusión

- El algoritmo secuencial es el que requiere de mayor tiempo de procesamiento, esto frente a las demás implementaciones debido a que estas utilizan las ventajas de la programación masivamente paralela.
- El uso de las máscaras de convolución en la memoria constante es una gran estrategia debido a que estas son utilizadas por todos los hilos durante la realización del filtro.
- Para imágenes pequeñas no es mucha la diferencia entre las diferentes técnicas, los mejores resultados se notan cuando el número de píxeles es mayor.
- La optimización del código puede mejorar en gran medida el tiempo de ejecución del algoritmo.
- Si no se libera el espacio reservado de memoria, en posteriores ejecuciones del algoritmo quedará datos guardados que afectarán el resultado final de la imagen.

#### 5. Trabajos futuros

- Buscar una solución a los problemas de divergencia en la parte de lectura de memoria global en la

implementación de memoria compartida.

- Usar un arreglo unidimensional para implementación de memoria compartida para que los queden contiguos y mejore el desempeño.
- Minimizar el número de ciclos en las implementaciones.

#### Referencias

- [1] INSTITUTO LATINOAMERICANO DE LA COMUNICACIÓN EDUCATIVA,*Procesamiento de imágenes*, la ciencia para todos, [Online], 2005
- [2] ESQUEDA ELIZONDO, J.J y PALAFOX MAESTRE, L.E,*Fundamentos de procesamiento de imágenes*, Universidad autónoma de baja California, 2005
- [3] GÓMEZ, J.C,*Introducción al procesamiento de digital de imágenes*, Universidad nacional de la plata, 2001
- [4] ALVAREZ BORREGO, J. *Aplicaciones del procesamiento de imágenes*, Centro de Investigación Científica y de Educación Superior de Ensenada, B.C, 2012
- [5] PICCOLI, M.F,*Computación de alto desempeño GPU*, Universidad nacional de la plata, 2011
- [6] BARRIUSO, J.*Aplicaciones del proceso digital de imágenes en Arqueología: Experiencias con los sistemas MIP y GEO-JARS de MICROM*,MICROM España, S.A, 1991
- [7] NVIDIA,*Procesamiento paralelo CUDA*, nVidia Corporation, 2017
- [8] SANCHIS, E,*Fundamentos y electrónica*, 2004