

**CONVOLUTION AND SOBEL FILTER SEQUENTIAL/PARALLEL WITH CUDA C AND
OPENCV**

Presentado por:

ANDERSON ALBERTO OCHOA ESTUPIÑÁN



**Presentado a:
JOHN OSORIO RIOS**

**UNIVERSIDAD TECNOLÓGICA
Marzo 11 del 2015
PEREIRA**

Introducción

En el siguiente trabajo se busca mostrar el comportamiento del filtro de Sobel (También conocido como filtro de detección de bordes) aplicado a diferentes imágenes, como también la implementación del mismo de manera paralela usando CUDA C y las diferentes formas de uso de memoria en la GPU vistas hasta el momento en el curso.

Para resolver el parcial se utilizó un equipo con las siguientes características principales (Por medio de la plataforma dispuesta para la compilación y ejecución de algoritmos en CUDA C y OpenCV):

CPU

- processor : 1
- vendor_id : GenuineIntel
- cpu family : 6
- model : 58
- model name : Intel® Core™ i7-3770K CPU @ 3.50GHz
- stepping : 9
- cpu MHz : 1600.000
- cache size : 8192 KB
- cpu cores : 4
- RAM: 16000mb

GPU

- Tesla K40c: 3.5
- Global memory: 11519mb
- Shared memory: 48kb
- Constant memory: 64kb
- Block registers: 65536
- Warp size: 32
- Threads per block: 1024
- Max block dimensions: [1024, 1024, 64]
- Max grid dimensions: [2147483647, 65535, 65535]

Requerimientos del parcial

1. Crear un programa que cargue una imagen desde un archivo, la lleve a escala de grises y realice un filtro de Sobel sobre esta imagen.
2. Se deben crear dos imágenes de salida:
 - a. Una es la imagen donde se resaltan los bordes de la imagen inicial una vez se le ha pasado el filtro de Sobel secuencial.
 - b. La otra es la imagen donde se resaltan los bordes una vez se le ha pasado el filtro de Sobel en paralelo.
3. Se deben crear tres programas distintos, que muestren la implementación de la convolución usando sólo memoria global, usando memoria constante y usando memoria compartida.
4. Se deben tomar tiempos de ejecución de cada algoritmo con imágenes de diferentes tamaños, las imágenes a usar estarán disponibles en el servidor (img1.jpg, img2.jpg, img3.jpg, img4.jpg, img5.jpg y img6.jpg).
5. Gráficas y conclusiones.

Algoritmos

Los algoritmos utilizados para dar solución al parcial se encuentran en:

<https://github.com/aaochoa/Hpc/tree/master/2D%20convolution>

Imagenes utilizadas y resultado obtenido

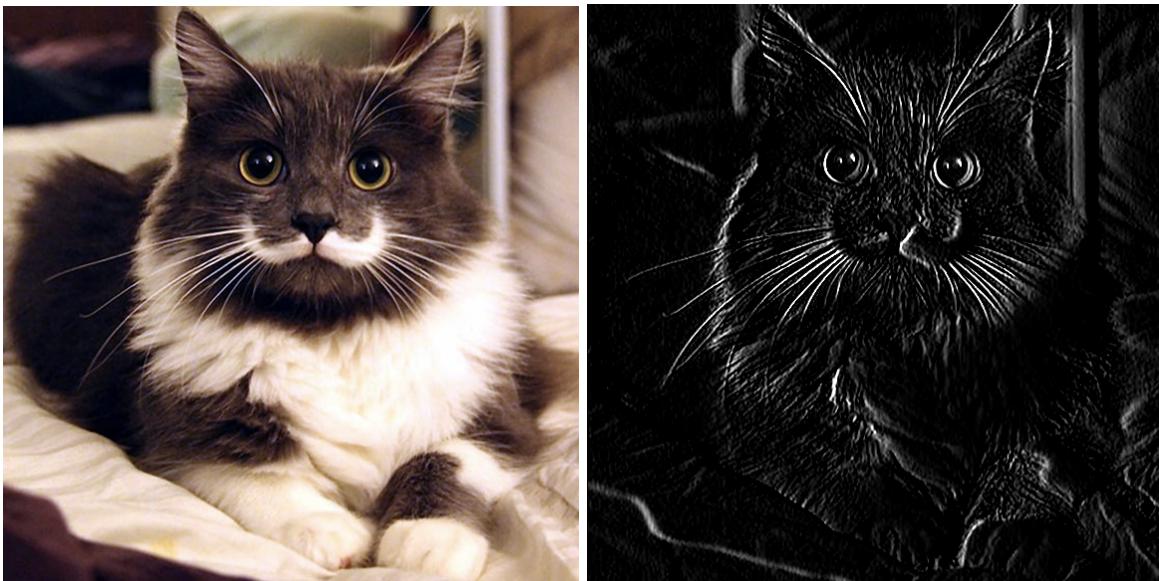


Imagen 1

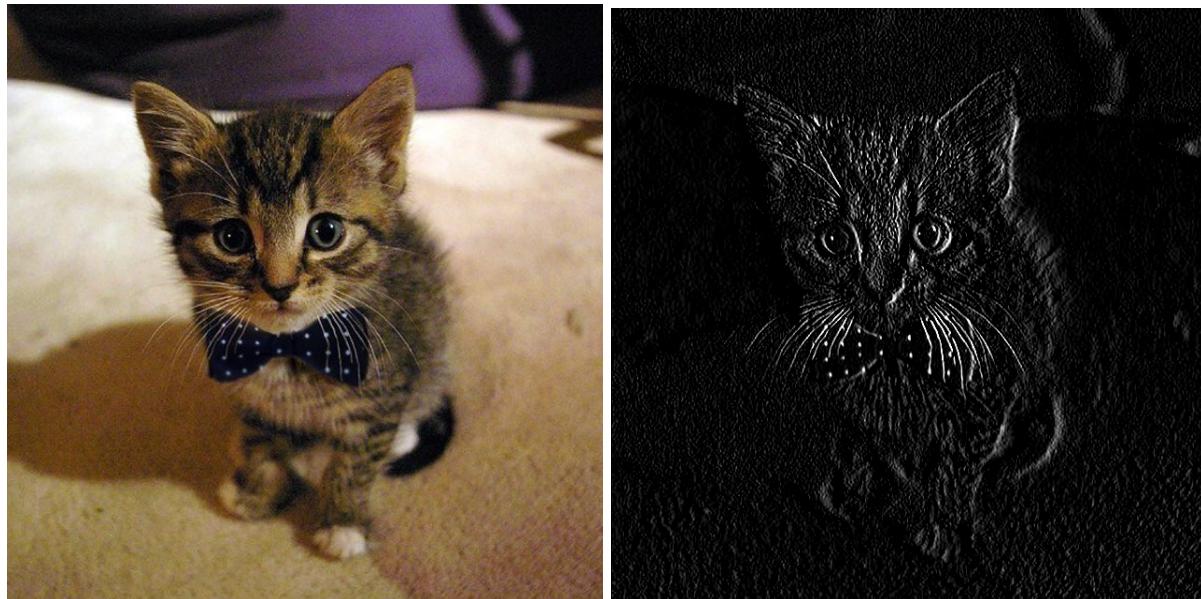


Imagen 2

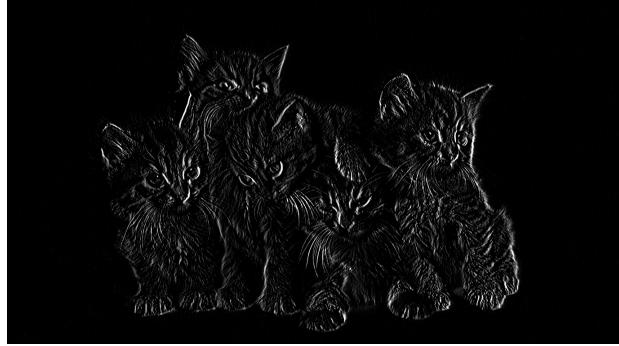


Image 3

Para observar el resto de las imágenes y sus resultados visitar el repositorio.

Tamaños de las imágenes utilizadas

Imagen 1: 580x580 pixeles

Imagen 2: 638x640 pixeles

Imagen 3: 1266x768 pixeles

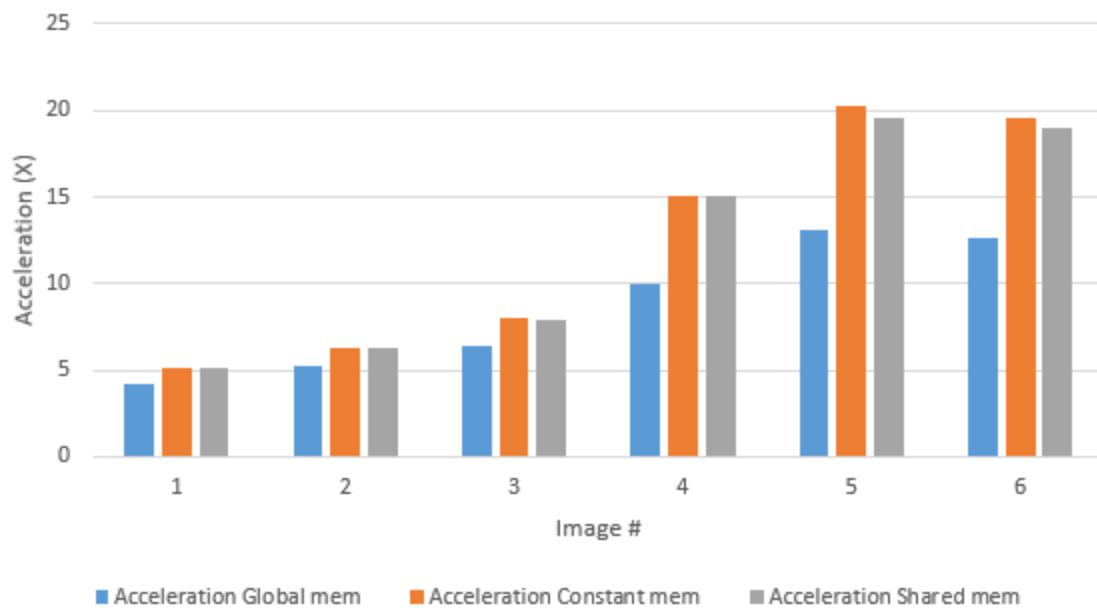
Imagen 4: 2560x1600 pixeles

Imagen 5: 5226x4222 pixeles

Imagen 6: 4928x3264 pixeles

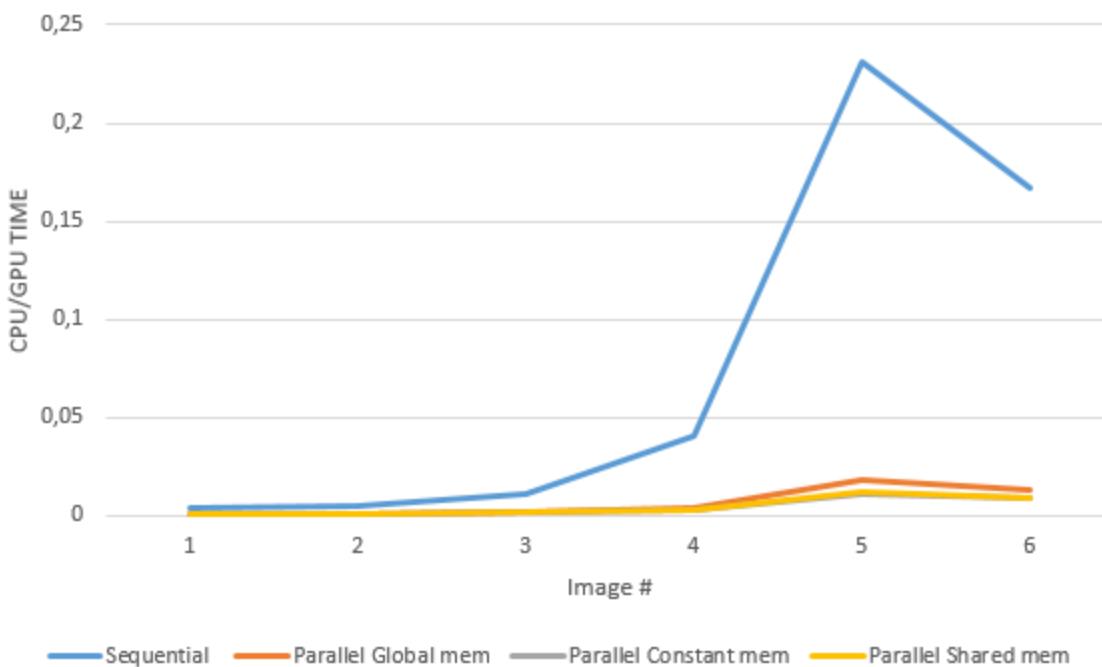
Gráficas y conclusiones

Acceleration Comparison

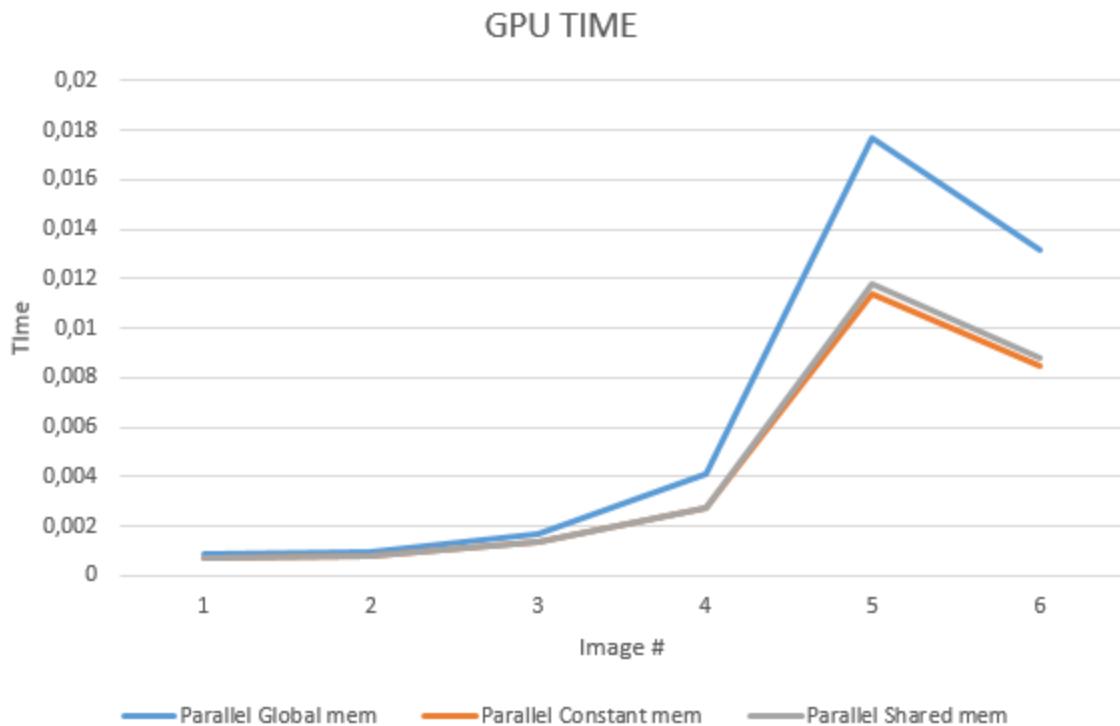


- La gráfica muestra el comportamiento de la aceleración de los 3 algoritmos dejando ver que en este caso la mejor versión es la que hace uso de memoria constante exclusivamente debido al paso de los datos de la imagen a memoria compartida.

CPU VS GPU TIME



- En la gráfica se ratifica la ventaja de hacer uso de versiones paralelas a la hora de manejar grandes volúmenes de datos (del orden de los millones).



- Visto en la gráfica la versión paralela que solo hace uso de memoria constante arroja tiempos de ejecución menores.
- En esta implementación no se han tenido en cuenta problemas tales como conflictos de bancos de memoria e hilos que quedan en estado idle.
- Las aceleraciones obtenidas con los algoritmos paralelos tienden a ser mayores conforme se aumente el tamaño de la imagen siempre y cuando los tamaños no superen los permitidos por las características de la tarjeta utilizada.
- Debido al tamaño de las imágenes la versión con memoria compartida fue aumentando sus tiempos de ejecución con respecto a los tiempos de la versión con memoria constante que lo hizo pero en menor cantidad.