Optimización del operador de sobel mediante computación masivamente paralela

Luis David Giraldo Grajales, *Estudiante, UTP,* Juan Sebastian Herrera Giraldo, *Estudiante, UTP*

***Abstract*—En este artículo se realizará una comparativa para establecer las diferencias de rendimiento en términos de tiempo de ejecució, entre la implementación del filtro de sobel disponible en la librería OpenCV y tres implementaciones del mismo realizadas para GPU usando diferentes técnicas de uso de memoria. Las implementaciones del filtro de sobel realizadas por el equipo están soportadas sobre la librería CUDA para dispositivos NVIDIA.**

***Keywords*—Filtro de sobel, Computación masivamente paralela, procesamiento de imágenes, memoria compartida, memoria global, memoria constante.**

I. Introducción

La idea de la computación de alto desempeño y los primeros dispositivos desarrollados con este fin aparecieron en la década de 1970. estos dispositivos han ido evolucionando dejando de ser simplemente una computadora con mayor capacidad, a convertirse en clusters basados en procesadores masivamente paralelos con interconexiones especiales, que les permiten aprovechar de mejor manera sus capacidades de procesamiento.

Con la aparición de dispositivos capaces de procesar grandes cantidades de información de manera paralela, aparecen técnicas de programación para aprovechar todas estas capacidades. la computación de alto desempeño ha facilitado la realización de muchas tareas de forma concurrente tales como: el proceso de tratamiento a imágenes (escala a grises, efecto blur, filtros de sobel), simulaciones por computadora (modelos climáticos,modelos evolutivos,modelos de sistemas planetarios), cálculos matemáticos complejos, el cálculo de la secuencia del genoma humano, entre otros.

Como podemos ver la tendencia en computación de alto desempeño basada en procesamiento masivamente paralelo es ideal para problemas de procesamiento de imágenes en los cuales se necesitan realizar una gran cantidad de cálculos independientes, por lo cual podría ser acertado prever que las implementaciones de este tipo tendrán un rendimiento mayor a las implementaciones secuenciales. Para demostrar esta hipótesis es necesario disponer de una implementación secuencial la cual pueda ser comparada mediante la toma de medidas de tiempo de ejecución con implementaciones paralelas. Las implementaciones paralelas será desarrolladas haciendo uso de la tecnologaía CUDA disponible para dispositivos NVIDIA.

Los dispositivos disponibles en la actualidad cuentan con diferentes tipos de memoria, cada una con unas características diferentes que las hacen tener mejores rendimientos en algunas tareas. Con el fin de aprovechar las capacidades de algunos tipos de memoria y comparar el impacto de estas sobre el rendimiento se realizarán tres implementaciones diferentes del filtro de sobel.

II. Procesamiento de las imágenes

El objetivo del procesamiento de las imágenes que realizaremos es aplicar el operador de sobel para resaltar o detectar los bordes en las imágenes de prueba, previo a aplicar el operador de sobel se representarn las imágenes a tratar en escala de grises con el fin de aplicar el operador de sobel sobre un solo canal de color.

A. Representación en escala de grises

En computación una escala de grises es una escala empleada en la imagen digital en la que el valor de cada píxel posee un valor equivalente a una graduación de gris.

Para representar una imagen en escala de grises que previamente estaba representada en píxeles rgb debe pasar por un proceso de transformación en el cual a cada píxel de la imagen original se le realiza una operación para encontrar su representación en escala de grises, para esta implementación se ven los valores rgb como variables de una ecuación en la cual los coeficientes están definidos de esta manera:

$$R \times 0.3 + G \times 0.59 + B \times 0.11 = P_{gris}$$

Fig. 1. Imagen en su representación de tres canales de color

B. Operador de sobel

operador de Sobel se utiliza para la detección de bordes en una imagen. Se detecta dos tipos de bordes. los bordes horizontales y los bordes verticales.

Los bordes se calculan mediante el uso de la diferencia entre intensidades de los píxeles correspondientes de una imagen. Para esto se usa una matriz de convolución denominada kernel, la cual tiene unos valores determinados para detectar bordes,

Fig. 2. Resultado de la aplicación de la escala de grises sobre la Fig. 1.

la operación de modificación con la matriz de convolución se aplica a cada pixel para obtener la matriz resultante.

Todas las máscaras de derivados deben tener las siguientes propiedades:

- Signo opuesto debe estar presente en la máscara
- Suma de máscara debe ser igual a cero.
- Maás peso significa más de detección de bordes.

operador de Sobel nos proporciona dos máscaras, una para la detección de bordes en la dirección horizontal y otro para la detección de bordes en una dirección vertical. La imagen final se construye asignado a cada píxel el valor calculado con la siguiente ecuación.

$$G = \sqrt{Gx \times Gx + Gy \times Gy}$$

siendo G el valor del pixel a calcular, Gx el valor del pixel con la misma coordenada en la matriz convolucionada en vertical, Gx el valor del pixel con la misma coordenada en la matriz convolucionada en Horizontal.

Fig. 3. Resultado de la Fig. 1.2. después de aplicado de filtro se sobel.

III. Memorias

Las GPUs actuales cuentan con diferentes tipos de memoria, estos tipos de memoria cuentan cada uno con características diferentes y pueden ser usados para mejorar el rendimiento de algunas aplicaciones, teniendo en cuenta esto se realizó el desarrollo de tres implementaciones diferentes del filtro de sobel, en cada una de las cuales se utiliza de manera distinta la memoria.

A. Global

La memoria global es la principal del dispositivo la cual pese a tener un ancho de banda inferior y una latencia superior a la de las demás ya que se encuentra fuera del chip, tiene la mayor capacidad de almacenamiento. Debido a el alto costo de realizar consultas a la memoria global es recomendable reducir en lo posible el número de las mismas.

La primera implementación se realizará usando únicamente la memoria global del dispositivo En la cual se espera que gracias a la implementacin paralela se obtenga una mejora notable en el rendimiento respecto a la implementación secuencial disponible en OpenCV.

B. Constante

La memoria constante del dispositivo pese a no poder ser modificada constantemente y tener una capacidad de almacenamiento inferior la memoria global, ofrece unas velocidades de lectura muy altas.

la segunda implementación se realizará haciendo uso de la memoria constante del dispositivo para guardar las matrices de convolución y también hará uso de la memoria global para guardar las matrices que representan las imágenes de entrada y las respectivas salidas.

C. Compartida

La memoria compartida se encuentra dentro del chip por lo cual tiene altas velocidades de transferencia, la lectura de la memoria compartida es casi tan alta como la realizada a los registros siempre y cuando no se generen conflictos, la memoria compartida está diseáda en forma de 16 bancos de 1kB cada uno por lo cual su capacidad de almacenamiento es muy limitada y únicamente puede ser accedida por hilos pertenecientes a su mismo bloque.

Aprovechando las características de este tipo de memoria, se realizará una implementación en la cual se guardarán en ella submatrices de la representación del archivo a procesar, también se usará la memoria constante del dispositivo para guardar las matrices de convolucin.

IV. Resultados

A continuación se presenta una comparación de los tiempos de ejecución de cada una de las implementaciones.

Las medidas presentadas a continuación fueron realizadas con 10 imágenes distintas ordenadas de mayor a menor respecto al tamaño de las mismas, los valores de tiempo representados corresponden al cálculo de la media de un conjunto de 20 mediciones de tiempo hechas para cada una de las imágenes.

A. CPU vs GPU

B. Gráfica de la aceleración

V. Conclusiones

En los resultados se evidencia que el rendimiento de los algoritmos desarrollados aplicando métodos de procesamiento

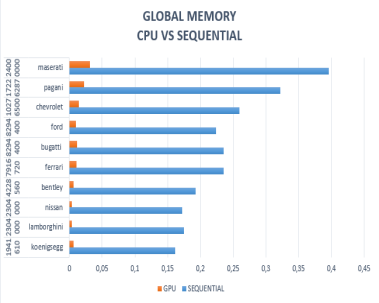

Fig. 4. Comparación de tiempos de ejecución de la libreria OpenCV y la implementación de memoria global realizada en GPU

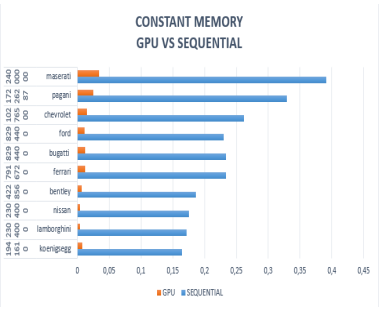

Fig. 5. Comparación de tiempos de ejecución de la libreria OpenCV y la implementación de memoria constante realizada en GPU

masivamente paralelo tienen un rendimiento superior en comparaciń con los disponibles en OpenCV, para la implementación basada en memoria global se observa una aceleracin de 26.38X respecto a la implementacin secuencial.

En la implementacin basada en memoria constante se esperaba tener una mejora en el rendimiento del algoritmo respecto a la implementación basada únicamente en memoria global pero los resultados nos arrojan que la aceleración respecto a la implementación secuencial es de 23.34X, y con respecto a la implementación basada en memoria global es de 0.88X. Esta pérdida de rendimiento puede deberse a la forma en la cual se asignó la memoria de tipo constante en el proceso de desarrollo.

En la implementación basada en memoria compartida se esperaba el mejor rendimiento de los tres algoritmos debido a que se usan los tres tipos de memoria en tareas tareas en las cuales arrojan sus mejores rendimientos. la aceleracin obtenida respecto a la implementación secuencial es de 38.14X, la aceleración obtenida respecto a la implementación basada en memoria global es de 1.44X y la aceleración obtenida respecto a la implementación basada en memoria constante es de 1.63X.

Se observa que la implementación con mejor rendimiento es la realizada usando una combinación de los tres tipos de memoria. Por lo tanto podemos Concluir que la correcta utilización los diferentes tipos de memorias tiene un impacto directo y significativo sobre el rendimiento de las aplicaciones.

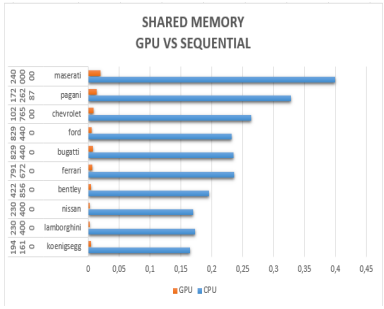

Fig. 6. Comparación de tiempos de ejecución de la libreria OpenCV y la implementación de memoria compartida realizada en GPU

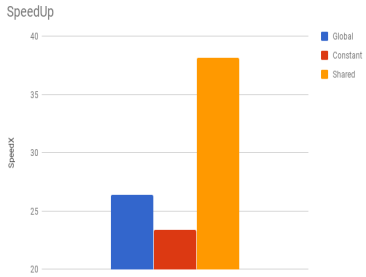

Fig. 7. Comparación entre las aceleraciones de memorias constante, global y compartida.

REFERENCES

[1] CARLOS JUEGA REIM ÚNDEZ, UCM., página [21, 26] *Estudio de rendmiento en GPU*, España.

[2] OPERADOR DE SOBEL, página [http://www.w3ii.com/es/dip/sobel_operator.html]

[3] DAVID B. KIRK, Wen mei W. Hwu, *Programming massively parallel processors*, 2010.