Detección de bordes usando CUDA

César Bragagnini Mendizábal*

Maestria en Ciencias de la Computación, Universidad Católica San Pablo, Arequipa, Peru

E-mail: cesarbrma91@gmail.com

Abstract

El siguiente trabajo muestra el performance al usar CUDA para el problema de Border Detection, se realiza una comparación con un programa de código serial(no paralelizado). Se uso para los experimientos imágenes de alta resolución.

Detección de Bordes

Para detectar los bordes se uso el Operador Sobel que utiliza dos kernels de 3x3 elementos para aplicar convolución a la imagen original calculando el valor absoluto del gradiente aproximando en cada pixel de la imagen en escala de grises, un kernel para los cambios en el eje horizontal y otro para el eje vertical. Aplicando a la imagen A por cada kernel, se obtiene las imagenes Gx y Gx

$$Gx = conv(A, \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}) \quad Gy = conv(A, \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix})$$

Estas imágenes Gx y Gx se puede combinar para hallar la imagen final G, donde el valor de cada pixel de G se obtiene por[1]

[1]Se puede obtener la combinación por otras formas por ejemplo usando formula pitagorica $|G_{ij}| = \sqrt{Gx_{ij}^2 + Gy_{ij}^2}$, usando el angulo de orientación $\theta_{ij} = arctan(Gx_{ij}/Gy_{ij})$, se eligio el operador suma de absolutos por ser mas fácil de calcular computacionalmente. [1]

$$G_{ij} = |Gx_{ij}| + |Gy_{ij}|$$

Calculando Operador Sobel por cada pixel

La imagen original debe ser convertida a escala de grises. Para obtener cada pixel por Gx y Gy, se hizo de la siguiente manera:

$$Gx_{ij} = conv(\begin{bmatrix} A_{i-1j-1} & A_{i-1j} & A_{i-1j+1} \\ A_{ij-1} & A_{ij} & A_{ij+1} \\ A_{i+1j-1} & A_{i+1j} & A_{i+1j+1} \end{bmatrix}, \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix})$$

$$Gy_{ij} = conv(\begin{bmatrix} A_{i-1j-1} & A_{i-1j} & A_{i-1j+1} \\ A_{ij-1} & A_{ij} & A_{ij+1} \\ A_{i+1j-1} & A_{i+1j} & A_{i+1j+1} \end{bmatrix}, \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix})$$

$$Gx_{ij} = (A_{i-1j+1} + 2A_{ij+1} + A_{i+1j+1}) - (A_{i-1j-1} + 2A_{ij-1} + A_{i+1j-1})$$

$$Gy_{ij} = (A_{i-1j-1} + 2A_{i-1j} + A_{i-1j+1}) - (A_{i+1j-1} + 2A_{i+1j} + A_{i+1j+1})$$

Resultados

Se uso una maquina con un procesador Intel Core I7, Memoria RAM de 8GB, Sistema operativo Ubuntu 14.04 de 64 bits y GCC 4.8.4 y con NVCC 6.5.12, con una tarjeta gráfica de GeForce GTX 750. Se implementaron 4 programas: 2 programas seriales que trabajan la imagen como una matriz 1d y el otro como una matriz 2d; otros 2 programas paralelos en CUDA, uno que usa la grid de bloques como 1d(solo eje x) y los threads en un bloque como 1d(solo eje x); y el otro programa en CUDA que usa la grid de bloques como 2d(eje x, eje y) y los threads en un bloque como 2d(eje x, eje y). Se usaron 6 imágenes de diferentes resoluciones. Table 1 muestra ***el tiempo en segundos al aplicar el operador Sobel a las imagenes***. Donde $N = numero\_filas\_imagen * numero\_columnas\_imagen/maxThreadByBlock$,

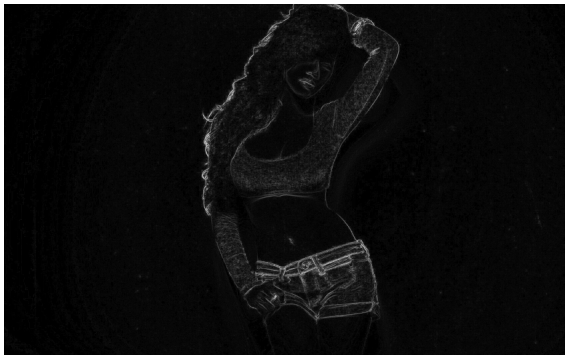
siendo $maxThreadsByBlock = 1024$, ademas Grid(MxN) significa que los bloques son organizados en una matriz de dimensión MxN, y Block(MxN) significa que los threads dentro de un bloque son organizados en una matriz 2d de dimensión MxN.

Table 1: Tiempo en segundos al aplicar el operador Sobel para detección de bordes

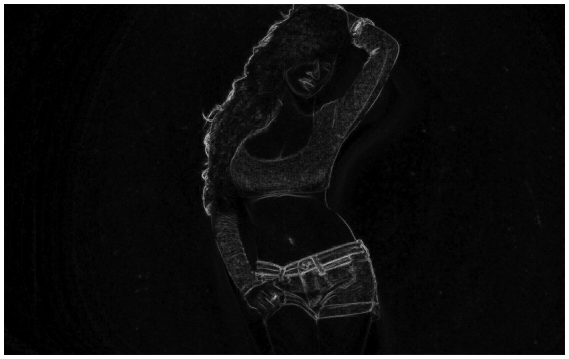
Res. Imagen	Serial 1d	Serial 2d	CUDA Grid1d Block1d	CUDA Grid(Nx10) Block(1024x1024)	CUDA Grid(Nx30) Block(1024x1024)	CUDA Grid(Nx50) Block(1024x1024)
3840x5760	0.769587	0.463506	0.074415	0.075105	0.073372	0.079719
4200x6720	0.955298	0.575857	0.102792	0.092168	0.093605	0.094402
5563x7981	1.541213	0.920442	0.143513	0.158117	0.148177	0.150842
5616x8600	1.659314	1.007712	0.161989	0.164865	0.165832	0.164255
6112x11658	2.430610	1.481908	0.251518	0.236386	0.232554	0.235345

Table 2: Imagén de 4200x6720 usada para aplicar detección de bordes

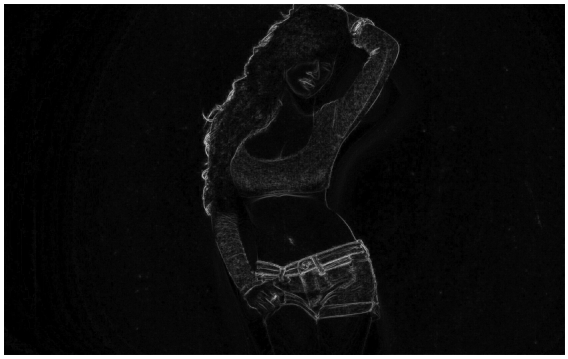
Imagen en escala de grises



Filtro serial 1d



Filtro CUDA 1d



Filtro CUDA Grid Block 2d

Conclusiones

- Debido a que la detección de bordes es un problema de naturaleza de operaciones de matrices es ideal para ser paralelizado usando CUDA, por esto todos los programas en CUDA tuvieron mejor performance que los programas seriales.
- No existe mucha diferencia en el rendimiento entre los programas CUDA que organizan a los bloques y threads como matrices 2d contra los que organizan a los bloques y threads como vectores.
- Los programas CUDA que organizán a los bloques y threads como matrices 2d de MxN, tendrìa mejor resultado si la elección de M y N seria la mínima, tal que permita que la cantidad de threads que están fuera de imagen sea lo menor posible.

References

(1) Fisher, R. Sobel Edge Detector. `http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm`.