Práctica 3: Difuminado de imagen CUDA

Andrés Fernando Román Arévalo, Ronald Sarmiento
Email:afromana@unal.edu.co, roasarmientoga@unal.edu.co
Universidad Nacional Bogotá, Colombia

***Resumen*—El siguiente informe presenta como se llevo a cabo la paralelización mediante el uso de la plataforma de programación en paralelo CUDA del efecto borroso y los resultados que se obtuvieron tanto de Speed Up como de Time Response para los distintas resoluciones de imagenes (720p, 1080p y 4k)) y de kernels (2,3,4,5, 6,7, 8,9, 10,11 ,12 13 y 14) asi como la comparación con los tiempos obtenidos en CPU**

I. Introducción.

Una de las tareas altamente demandada en la actualidad es el procesamiento de imagenes cada vez de una mayor resolución para lo cual se requiere un alto nivel de computo. A partir de esto han surgido innumerables algoritmos que permiten paralelizar las distintas operaciones de procesamiento de imagenes como lo es la del efecto difuminado.

Aun asi paralelizar tiene su limite que es explicado en la Ley de Amdahl, la cual establece que hay un punto en que paralelizar ya no ofrece mejoras significativas de rendimiento e incluso por el contrario puede llevar al deterioro del mismo.

Al momento de hacer procesamiento de imagenes se utiliza distintas operaciones matriciales.En el caso del efecto borroso se realiza una convolucion entre la matriz que representa la imagen y el kernel

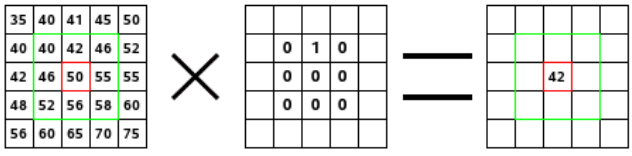


Figura 1: Convolucion matricial

I-A. Creación del efecto difuminado

Mas especificamente se utilizo la tecnica de difuminado conocida como Blur el cual consiste en intercambiar cada pixel de la imagen por un promedio de los valores RGB que se encuentras en los pixeles adyacentes a el. El nivel de difuminado esta determinado por el kernel y el tamaño da el rango que se toma para evaluar el promedio. Es decir, si se escoge un kernel de 5 significa que cada pixel sera cambiado por el promedio que hay en los promedios de la matriz 5x5 cuyo centro es el pixel en cuestion.

II. Algoritmo de paralización.

La paralelizacion se realizo mediante asignación tipo block-wise es decir que se tomo la imagen y se dividio a lo alto en filas y cada una de estas se asigno a un hilo. Es decir que un caso hipotetico donde se quiera aplicar el efecto difuminado a una imagen con un alto de 368 pixeles,se lanzan 4 hilos donde el hilo 1 se le asignaba la fila 1(del pixel 0 al 92), al 2 la fila 2(del pixel 92 al 184), al 3 la columna 3(del pixel 184 al 276) y al 4 la columna 4(del pixel 276 al 368) . Para mayor clarificación detras de la idea que se uso para la paralización, la siguiente imagen:

Figura 2: Algoritmo de paralelización

El ejemplo anterior muestra el caso en el cual se lanza un solo bloque con 4 hilos. Pero para el experimento se decidio lanzar la máxima cantidad de hilos posibles es decir el alto de la imagen para calcular tanto el time response como el speed up en base al tamaño del kernel

III. Experimentos y resultados.

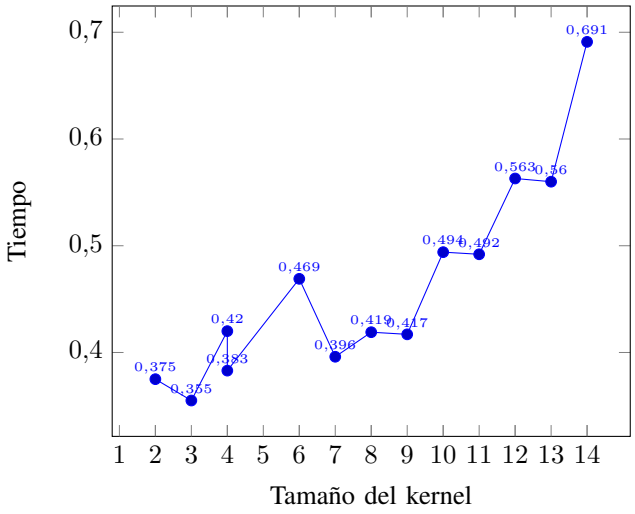
Antes de mostrar los resultados obtenidos y compararlos con los que se obtuvieron al paralelizar el efecto difuso que se obtuvo con CUDA en comparación con los de CPU(POSIX Y OpenMP), es necesario decir que se corrio en el entorno de Google Colab.

III-A. Resultados

Como primera medida se utilizo el time response o tiempo de ejecución de un programa, donde para cada imagen se crearon sus corresponientes gráficas donde en eje horizontal se colocaron el tamaño del kernel que ejecutaba el programa y en el eje vertical el tiempo que tardo.
A continuacion se presentan las gráficas correspondientes:

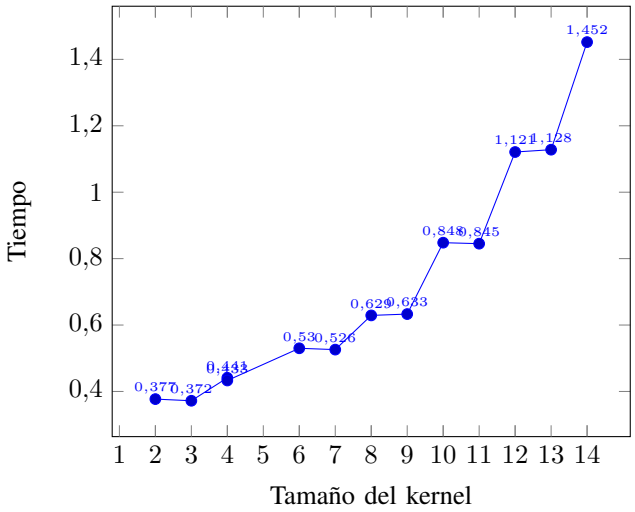
- **Imagen de 720p**

Time Response 720p



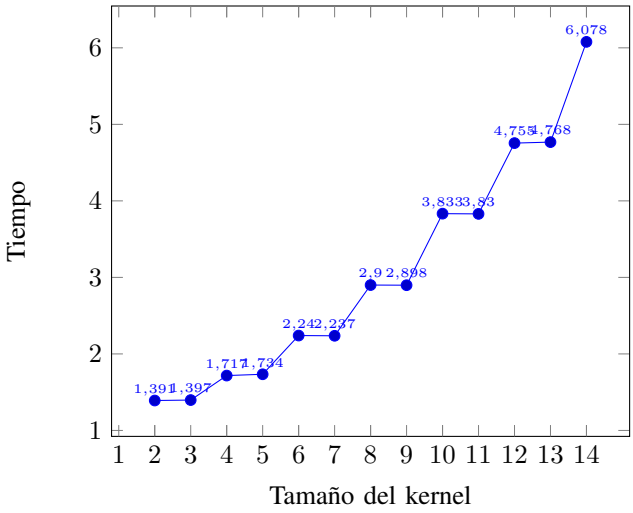
- **Imagen de 1080p**

Time Response 1080p



- **Imagen de 4k**

Time Response 4k

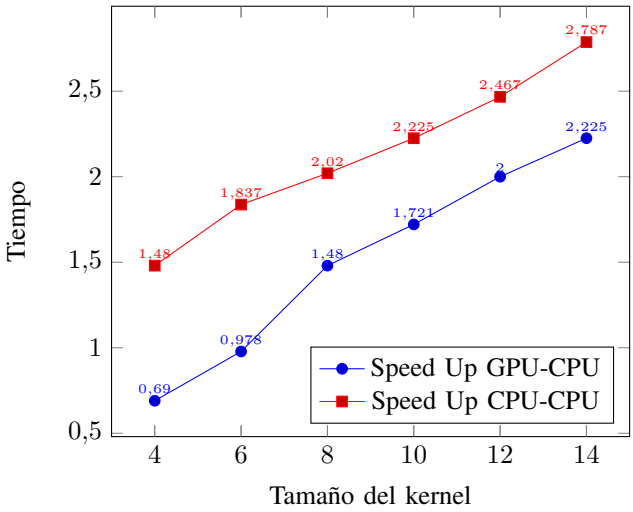


Como se puede observar en las gráficas anteriores el time response del código tiene cierta tendencia de empeorar conforme se aumenta el tamaño del kernel, como es el comportamiento esperado.

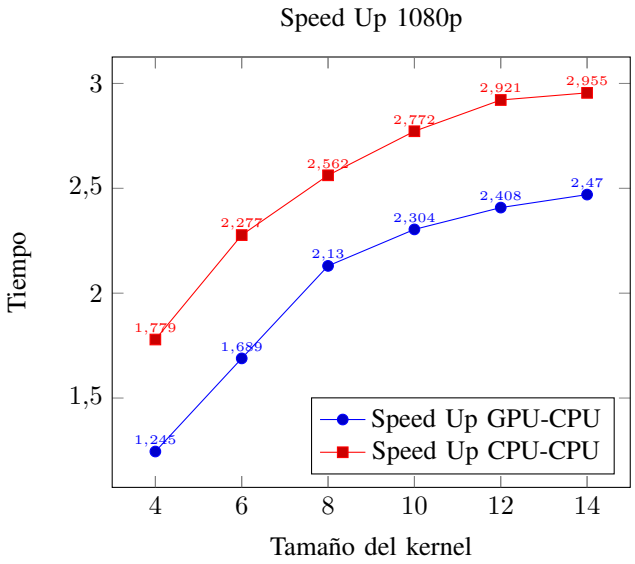
Otra medida que se considero determinar para medir el rendimiento de los hilos en el programa es el speed up que para este caso se calculo como la división entre el mejor tiempo en CPU contra con los mejores tiempos en GPU
A continuación se presentan las gráficas correspondientes:

- **Imagen de 720p**

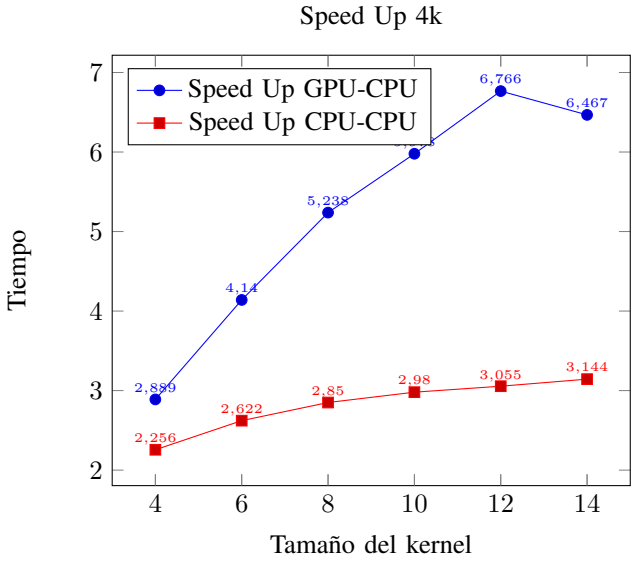
Speed Up 720p



- **Imagen de 1080p**



- **Imagen de 4k**



CONCLUSIONES.

- Entre mas grande es el kernel con el que se trabaja se demora un tiempo superior en realizar el efecto difuminado debido a que realiza mas operaciones al momento de calcular el promedio del kernel, haciendo que el costo computacional sea superior
- Se evidencia que el speedup de GPU es menor cuando el computo a realizar es considerablemente menor, ya que es necesario tener en cuenta los tiempos de copia de los datos a calcular en el device y luego la recepcion en host, además que para la implementacion en cuestión es necesario realizar una transformacion de matrizde OpenCV a arreglo de enteros.
- Es necesario realizar una comparacion de las implementaciones de CPU y GPU para reconocer cual tiene mejor rendimiento, para nuestra práctica solo vale la pena usar GPU en el computo de imagenes 4K.

REFERENCIAS

[1] Pedraza, C. (2019). Diapositivas de clase.

[2] David, K. and Wen-Mei, H. (2010). Programming massively parallel processos. 2nd ed. Morgan Kaufmann.

[3] Thomas, R. and Gudula, R. (2010). Parallel Programming. 1st ed. Springer.

[4] En.wikipedia.org. (2019). $Boxblur. Availableat : https : //en.wikipedia.org/wiki/Box_blur.$