Algoritmo Paralelizable en C para Difuminar Imágenes

Andres Duvan Chaves Mosquera
adchavesm@unal.edu.co
Juan Pablo Girón Bastidas
jpgironb@unal.edu.co

I. Resumen

Para la clase Computación paralela y distribuida, siguiendo con el estudio de la paralelización de procesos, vamos a estudiar un algoritmo de difuminación de imágenes. Como complemento al estudio de la paralelización vamos a analizar el tiempo de ejecución de este algoritmo, variando ciertos parámetros como el tamaño del kernel, la cantidad de hilos, y la cantidad de pixeles (resolución). Con esto calcularemos el rendimiento del programa bajo distintas condiciones y conseguiremos su speedup.

II. Diseño

El proceso de difuminación se llevó a cabo mediante convoluciones entre un kernel (matriz cuadrada de tamaño impar) y una vecindad de pixeles en toda la imágen. El tipo de difuminación es la Gaussiana ya que el kernel utilizado sigue una distribución normal para datos bidimensionales:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Figura 1. Distribución normal de datos bidimensionales.

A los datos en matriz se les aplica posteriormente un proceso de normalización para un difuminado más limpio y así no modificar la densidad de frecuencias de colores en la imágen.

Por otro lado, para la gestión de las imágenes nos apoyamos de una librería de terceros de C llamada stb library, la cual nos permite decodificar fácilmente los datos binarios contenidos en ellas, esta librería trabaja con los formatos de imágen más conocidos.

Para trabajar fácilmente con los códigos RGB de las imágenes usamos un vector de números enteros, el cual representa los valores de los tres canales por cada pixel.

Posteriormente, tenemos que aplicar la convolución a cada pixel de la imágen, y a su vez, cada uno de los tres canales RGB.

Además, trabajaremos usando un espacio de memoria compartido entre los hilos, que representa los valores RGB de los pixeles; un vector de enteros, de forma que cada hilo tenga conocimiento del puntero del primer elemento del vector. En esta situación, sin embargo, no tendremos que aplicar un paso de reducción posterior al procesamiento, ya que la lógica de nuestro programa no lo requiere. Adicionalmente, no tendremos que preocuparnos por posibles condiciones de carrera puesto que las convoluciones trabajan sobre un rango de píxeles bien definido en el cual no hay superposiciones.

La repartición del trabajo se basa en el enfoque blockwise, en donde cada hilo trabaja sobre un rango de píxeles de igual tamaño, como se puede notar en el diagrama:

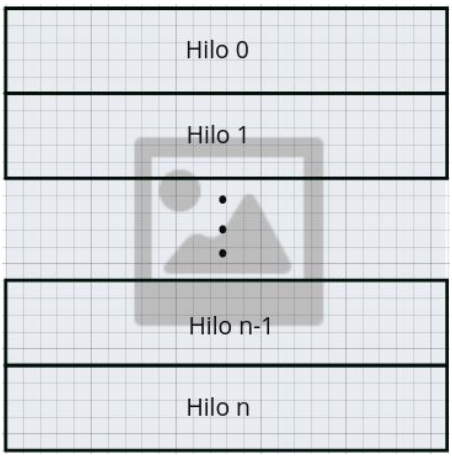


Figura 2. Repartición de trabajo según enfoque blockwise.

III. Experimentos

Variables: Para analizar el comportamiento de los hilos en la ejecución, vamos a variar 3 factores:

Matriz kernel, esta matriz es la que nos permite hacer el efecto de difusión, vamos a operar cada pixel de la imagen real con esta matriz mediante un algoritmo, este valor se variara en 4 valores distintos(3,7,9,15) tamaños posibles para hacer la difuminación.

Tamaño de la imagen: Ya que el algoritmo opera por pixeles de una imagen, variar la cantidad de píxeles nos sirve también para evaluar su comportamiento, para este caso vamos a probar con imagenes en resolucion HD, FHD Y 4K.

Número de Hilos: El número de hilos es la variante más importante, ya que mediante este es que veremos más evidente la reducción de tiempo a medida que más hilos colaboran para ejecutar el algoritmo.

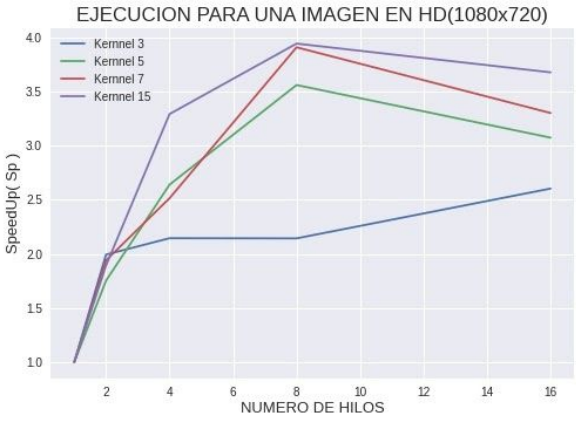
Una vez sabemos cuales son las variables de la ejecución vamos a experimentar con todas las posibles combinaciones y presentar los resultados en una gráfica. Y como el principal interés es graficar número de hilos con relación al tiempo que se demora, podemos

IV. Resultados

Gráfica 1. Ejecución del algoritmo para una imagen en resolución HD.Analisis Número de hilos vs tiempo.

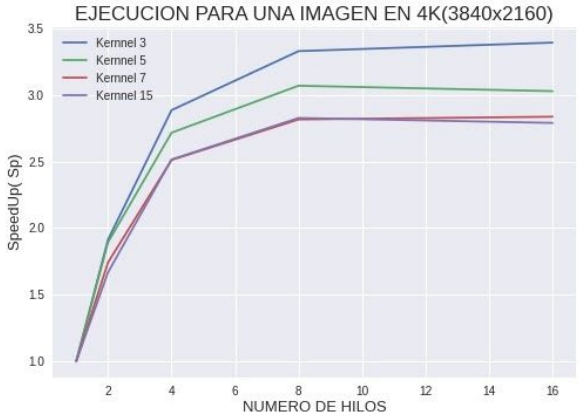
Gráfica 2. Ejecución del algoritmo para una imagen resolución FullHD. Analisis Número de hilos vs tiempo.

Gráfica 3.Ejecución del algoritmo para una imagen resolución 4K. Analisis Número de hilos vs tiempo.



Gráfica 4.Ejecución de el algoritmo para una imagen resolucion HD. Analisis Número de hilos vs SpeedUp.

Gráfica 5. Ejecución del algoritmo para una imagen resolución FullHD. Analisis Número de hilos vs SpeedUp.



Gráfica 6. Ejecución de el algoritmo para una imagen resolución 4K. Analisis Número de hilos vs SpeedUp.

V. Conclusiones

Inicialmente tenemos, las gráficas que comparan Tiempo de ejecución contra número de hilos, de estas gráficas podemos inferir muchas cosas, primero que todo, se evidencia que las secciones de código que paralelismos representan casi toda la ejecución esto se puede evidenciar por la curva de la gráfica.

La matriz kernel solo aumenta la cantidad de operaciones por lo que la tendencia de las curva se comporta casi igual en todas las gráficas sin importar el tamaño de la matriz kernel.

Algo bastante importante de notar es que a partir de 4 hilos, el tiempo no mejora considerablemente. Esto sucede ya que el computador en el que se llevó a cabo los experimentos posee 4 cores físicos y 4 virtuales. Esto también se evidencia en las gráficas de speedup, ya que la curva tiene el mismo comportamiento, hasta 4 hilos se evidencia el cambio y después se mantiene.