Computación paralela y distribuida
Práctica 1

Sebastian Guerrero Salinas sebguerrerosal@unal.edu.co
Juan Camilo Monterrosa Sanchez jcmonterrosas@unal.edu.co
Guiselle Tatiana Zambrano Penagos gtzambranop@unal.edu.co
Universidad Nacional de Colombia
Bogotá, Colombia, Abril 2 de 2020

Resumen

El presente es el reporte de la primera práctica en la asignatura "Computación paralela y distribuida", práctica que consiste en la implementación y comparación del rendimiento de un algoritmo que genere un efecto borroso sobre una imagen. Este algoritmo se ejecutará en dos escenarios distintos con sus respectivas instrucciones, y sobre cada uno de ellos se realizarán las pruebas pertinentes con su respectivo informe de resultados.

Palabras clave: Efecto borroso, hilos, imagen, paralelización, tiempos de respuesta.

I. Introducción

Este documento es ofrece una comparación clara y concluyente del rendimiento de un algoritmo para la generación del efecto borroso en una imagen en escenarios diferidos uno del otro por las condiciones de paralelización, el primero de ellos en una ejecución secuencial y el segundo en una ejecución con diferentes cantidades de hilos POSIX. Cada uno de los escenarios de ejecución se probará con tres tipos de imágenes y el algoritmo de generación de efecto borroso será el mismo, una vez ejecutado cada escenario se tendrá una imagen resultante de la aplicación del efecto y se realizará la toma de medidas de tiempo de respuesta para cada imagen y se llevará el registro de las mismas.

II. Procedimiento

1. *Elaboración y prueba del algoritmo:*

   Para la construcción del algoritmo generador del efecto borroso se tomó como base varias implementaciones del algoritmo "Fastest Gaussian blur" [1], en esta interpretación se definía el efecto Gaussian Blur de una función en dos dimensiones como la convolución de esa función con funciones Gaussianas en dos dimensiones. En este caso particular de la función Gaussiana adaptada se maneja una integral que está únicamente definida por un parámetro σ llamado desviación estándar y que para fines prácticos de comprensión del algoritmo lo notan como "radio". Teniendo un caso discreto finito presentan las funciones en dos dimensiones como matrices de valores y se computa el volumen (integral) como una suma, como la función Gaussiana tiene valores muy cercanos a cero en ciertos radios, se usarán solo los valores $-r \leq x \leq r$, $-r \leq y \leq r$, es acá donde se empieza a manejar el kernel, interpretado como la parte "útil" del peso. Así el valor de convolución en la posición $[i, j]$ es el promedio ponderado, es decir, la suma de los valores de la función alrededor de $[i, j]$ multiplicado por el peso.

   En el desarrollo de este reporte se decidió estudiar y adaptar una de las 4 implementaciones del "Fastest Gaussian blur", más específicamente la llamada "gaussBlur_3" por el autor. Esta implementación define dos grandes funciones que terminan produciendo un "desenfoque unidimensional", desenfoque el cual se describe como un desenfoque de caja y permite visualizar de mejor manera que las dos grandes funciones que lo producen tienen una complejidad de $O(n * r)$, complejidad heredada de dicho desenfoque de caja.

   El programa funciona dividiendo la imagen en los canales que la conforman, para este caso particular en el que nos encontramos trabajando sobre extensiones **.jpg** y **.jpeg**, se tienen los canales **R**, **G** y **B**, los cuales de manera secuencial son ingresados individualmente a la función generadora del algoritmo de **Blur**, ahora bien, para lograr la paralelización, se optó por repartir los datos de tal manera que las funciones correspondientes que modifican la información de cada canal dividieran el proceso por la cantidad de hilos creados, utilizando así una **Distribución de datos** de tipo **BlockWise**, como se representa a continuación:

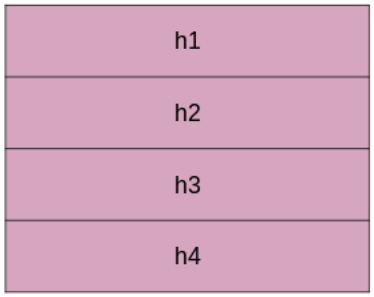

Figura 1. Block-wise

2. *Construcción del programa secuencial:*

   Al analizar el problema buscando la mejor solución se logró dar con una librería de manejo de imágenes con soporte y buena documentación para el lenguaje de programación a utilizar el cual es **C**, la librería en mención es **STB** [2] y para poder hacer uso de la misma se debía descargar en el mismo directorio donde se tuviera el programa. En cuanto a la estructura del programa se decidió empezar con la inclusión de las librerías que fueron más exactamente 7, seguido de ello se incluye una macro de utilidad para verificación de errores, luego se define una estructura y varias funciones de comparación entre números, le siguen funciones de manejo de imágenes y posterior a ello todas las funciones que componen el algoritmo de efecto borroso; para el *main* del código se dispondrá de sentencias que asemejan a un programa paralelizado con hilos POSIX pero para la disposición particular de este programa en forma secuencial solo se manejará con un hilo, lo cual refleja un comportamiento idóneo para este escenario. La organización del *main* consta de una verificación inicial de errores para el comando que ejecute el programa, seguido se encuentra la asignación del número de hilos, luego recibe el número que indica el tamaño del kernel, posteriormente se carga la imagen y le siguen las sentencias de manejo de hilos que, para el caso de este programa sería únicamente uno, finalmente termina guardando la imagen resultante y liberando la memoria de esta.

3. *Construcción del programa con hilos POSIX:*

   La estructura del escenario de paralelización con hilos POSIX es en gran medida muy similar a la estructura

del programa para el escenario secuencial ya que la estrategia de construcción del mismo fue hacer un mismo estilo de programa que funcionara para ambos casos con la diferencia que para el caso de la paralelización se varía el número de hilos a ejecutar el programa y en el secuencial se deja este número constante con un valor de 1.

4. *Creación del script de ejecución total*

Como herramienta para la ejecución y prueba de los programas en este reporte se solicitó la construcción de un script de ejecución total, es decir, un archivo de texto cuyo contenido es un conjunto de líneas de comando, las cuales son ejecutadas de forma secuencial de principio a fin como si se estuvieran realizando de forma manual en la terminal de consola del sistema operativo que para este reporte es uno basado en Linux, más específicamente **Ubuntu 18.4**.

El script creado y utilizado para esta ocasión es llamado **"script_ejecutar_todo.sh"** y está totalmente enfocado a la compilación y ejecución de los programas a evaluar, a la variación de los parámetros de ejecución de cada uno y la recolección de resultados de tiempo de cada uno en archivos de texto plano. La estructura del script se resume a continuación:

a) *Compilación del programa:* dos líneas de comandos que cumplirán la función de compilar los programas a evaluar, estas líneas con los respectivos parámetros de funcionamiento de las librerías que se usan en los archivos y la especificación del nombre del ejecutable.

b) *Resultados del programa en secuencial:* esta parte del script comienza con la creación del archivo llamado **"resultados.txt"** el cual será el archivo donde se colectarán los resultados. Seguido de eso, comienza la parte del registro de datos para el programa que trabaja de forma secuencial, se realiza con caracteres sencillos una figura de tabla con encabezado que dice **"Ejecución secuencial"**, seguido de ello se indica el valor inicial del kernel que será de 3, luego, un separador que indica que se mostrarán los resultados de la primera imagen, la imagen de 720p, después un condicional que indicará el tamaño de kernel con el que se está trabajando en el archivo de resultados seguido de la línea de comando que ejecuta el programa especificando el nombre del ejecutable y la imagen que va a tratar el programa y toma el tiempo de dicha ejecución, finaliza esta parte con un incremento en el número de kernel en 2, todas las instrucciones del condicional están destinadas a repetirse mientras el kernel no supere el valor de 15; todas estas instrucciones se repiten para las imágenes de 1080p y 4k respectivamente.

c) *Resultados del programa en paralelo:* esta parte del script comienza con la parte del registro de datos para el programa que trabaja de forma paralela con hilos POSIX, los datos acá registrados irán de igual manera al archivo **"resultados.txt"**, se realiza con caracteres sencillos una figura de tabla con encabezado que dice **"Ejecución en paralelo"**, seguido de ello se indica el valor inicial del kernel que será de 3 y el valor inicial de cantidad de hilos que será de 2, luego, un separador que indica que se mostrarán los resultados de la primera imagen, la imagen de 720p, después dos condicionales, uno dentro del otro con el propósito de ir aumentando el tamaño de kernel para el condicional externo e ir aumentando la cantidad de hilos para el condicional interno, sus instrucciones son indicar y grabar en el archivo de resultados el tamaño de kernel y la cantidad de hilos con las que se ejecuta el programa cada vez, le sigue la línea de comando que ejecuta el programa especificando el nombre del ejecutable y la imagen que va a tratar el programa y toma el tiempo de dicha ejecución, finaliza esta parte con un incremento en el número de kernel en 2 y multiplicando la cantidad de hilos por 2, todas las instrucciones de los condicionales están destinadas a repetirse mientras el kernel no supere el valor de 15 y la cantidad de hilos se mantenga menor a 17; todas estas instrucciones se repiten para las imágenes de 1080p y 4k respectivamente.

III. Resultados

El computador utilizado para realizar la ejecución de la práctica y todas las pruebas cuenta con un procesador AMD A8 - 7410 con 2.2 GHz, tiene una memoria RAM de 4GB y un disco duro con una capacidad de 500GB.

Para obtener los datos, fue necesario ejecutar 10 veces el script llamado *script_ejecutar_todo.sh* y los valores arrojados por este fueron manejados en un notebook en Colab [2] en donde fueron promediados y clasificados de la siguiente manera:

- **Kernel 3**

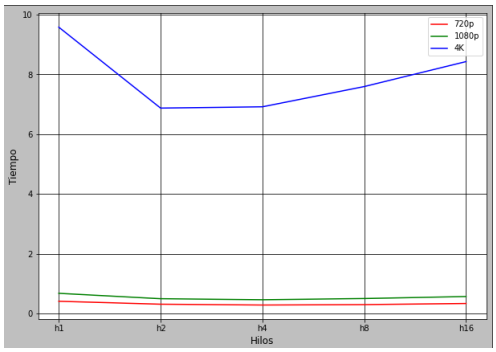

Figura 2. Tiempo vs Hilos

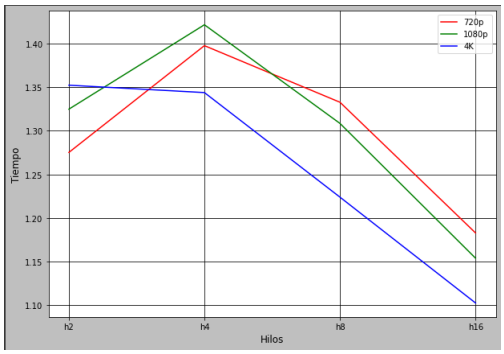

Figura 3. Speed-up

- **Kernel 5**

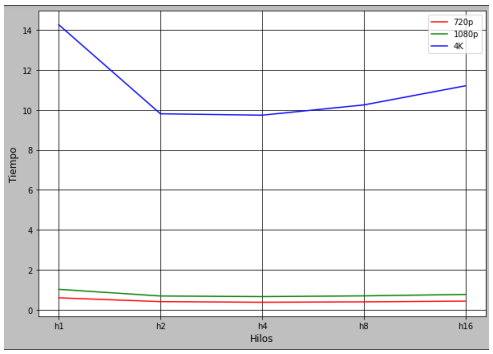

Figura 4. Tiempo vs Hilos

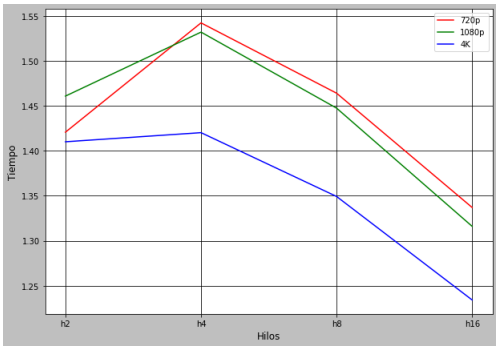

Figura 5. Speed-up

- **Kernel 7**

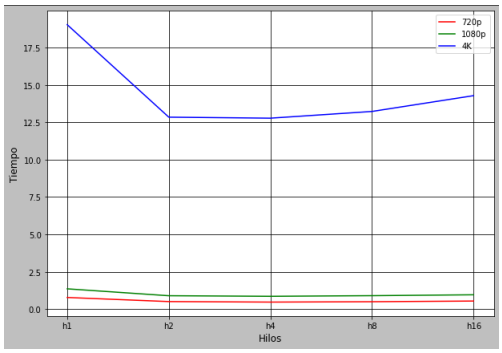

Figura 6. Tiempo vs Hilos

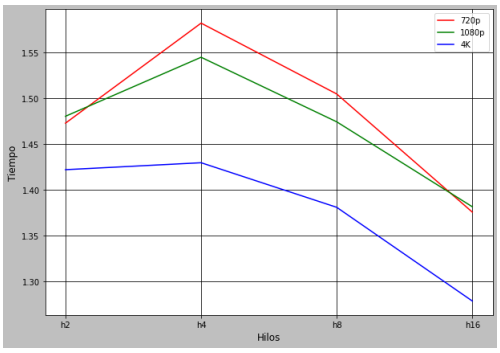

Figura 7. Speed-up

- **Kernel 9**

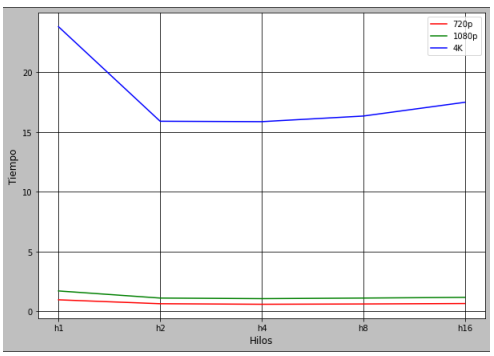

Figura 8. Tiempo vs Hilos

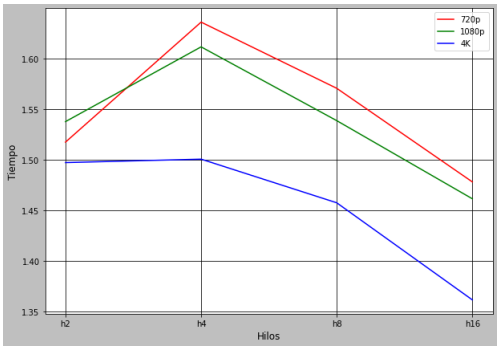

Figura 9. Speed-up

- **Kernel 11**

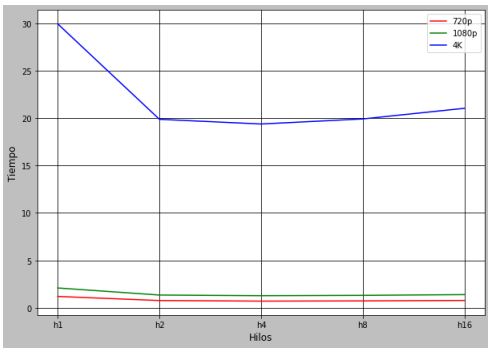

Figura 10. Tiempo vs Hilos

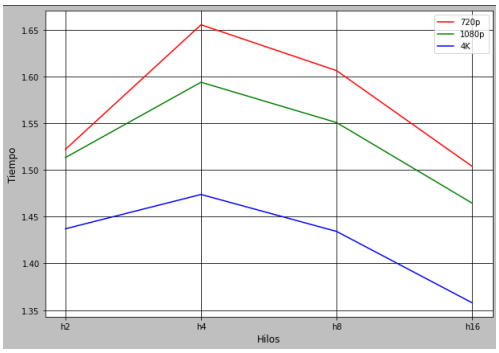

Figura 11. Speed-up

- **Kernel 13**

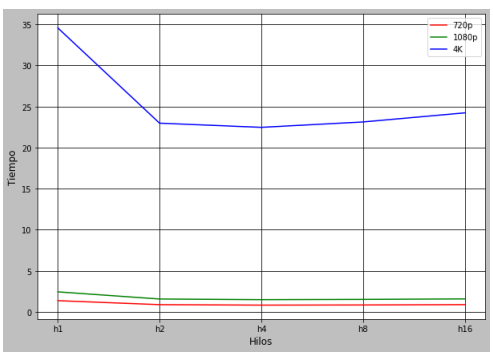

Figura 12. Tiempo vs Hilos

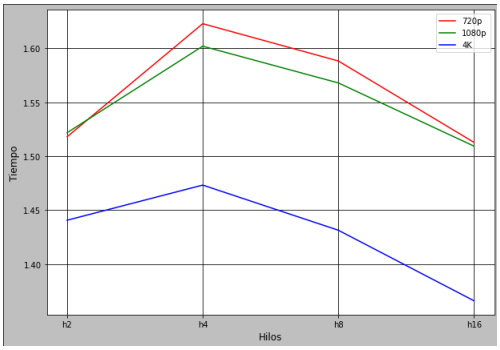

Figura 13. Speed-up

- **Kernel 15**

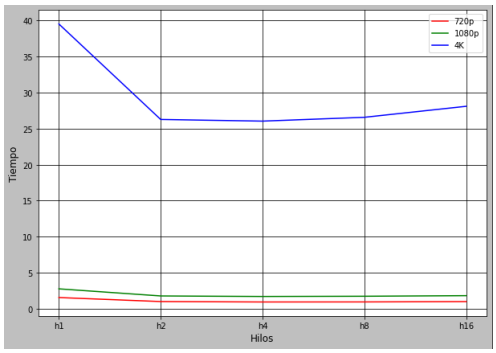

Figura 14. Tiempo vs Hilos

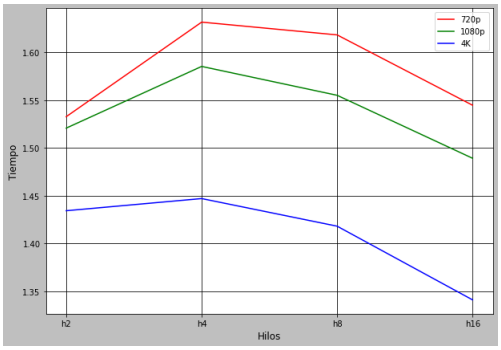

Figura 15. Speed-up

IV. Conclusiones

- Al incrementar el tamaño del kernel el efecto blur es más notable en las imágenes
- A causa de la cantidad de píxeles, con números de kernel grandes y bastantes hilos, la calidad del efecto borroso disminuye notoriamente
- Únicamente la imagen de 4k amerita realmente incrementar el tamaño del kernel y aumentar los hilos ya que resalta más la optimización de tiempo de ejecución del programa
- Mediante el *Speed-up* podemos notar el efecto que tienen los núucleos de procesamiento de un computador en la programación paralela, puesto que en todas las diferentes ejecuciones variando los *kernels* y la cantidad de hilos utilizados, el máximo *Speed-up* es 4, que resulta ser igual al núumero de núcleos del computador donde se esta ejecutando el programa.

Referencias

[1] Fastest Gaussian Blur (in linear time). Available at: http://blog.ivank.net/fastest-gaussian-blur.html https://github.com

[2] Notebook Colab. Available at: https://colab.research.google.com/drive/1WvBwBzceWoZUzsvXcfuIBtCEchWDNFA6