Computación Paralela y Distribuida
Practica # 1

José Fernando Morantes Florez, Johan Sebatian Salamanca Gonzalez, Harold Nicolas Saavedra Alvarado
{jfmorantesf, jssalamncag, hnsaavedraa}@unal.edu.co

I. Algoritmo de bluring

Se utilizo como base el algoritmo de **Gaussian Blur** la cual se explica en la siguiente imagen.

$$b[i,j] = \sum_{y=i-r}^{i+r} \sum_{x=j-r}^{j+r} f[y,x] * w[y,x]$$

Sin embargo, para la construcción de un algoritmo mas eficiente se utiliza el box blur el cual al realizarlo repetidas veces (en nuestro caso consideramos 3 veces) sobre la imagen se aproxima al resultado que se obtiene utilizando gaussian blur. Para este algoritmo se utiliza la siguiente constante para los pesos

$$\frac{1}{2 * br^2}$$

Donde br [2] es el radio ideal de la caja. Teniendo esto en cuenta se obtiene la siguiente formula.

$$bb[i,j] = \sum_{y=i-br}^{i+br} \sum_{x=j-br}^{j+br} f[y,x]/(2 \cdot br)^2$$

Como podemos observar esta formula se aplica a cada pixel de la imagen y la idea general es realizar el efecto de blur utilizando los pixeles vecinos simulando un suavizado de estos.

Para hacer este algoritmo mas eficiente se dividio la operación de convolucion en 2, en la primera de estas (Horizontal Blur) se itera sobre las columnas dejando estática la fila y se alplica la siguiente formula.

$$b_h[i,j] = \sum_{x=j-br}^{j+br} f[i,x]/(2 \cdot br)$$

Luego, sobre el resultado del Horizonta blur, se itera sobre la fila y se deja estática la columna aplicando la formula mostrada a continuacion, a esto le llamamos Total blur.

$$b_t[i,j] = \sum_{y=j-br}^{j+br} b_h[y,j]/(2 \cdot br)$$

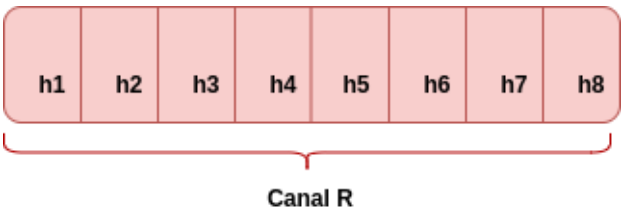
Se demuestra entonces que al aplicar estos dos procedimientos sobre la imagen se obtiene el mismo resultado e incluso la misma formula de box blur, pero con una complejidad de $O(n * r)$

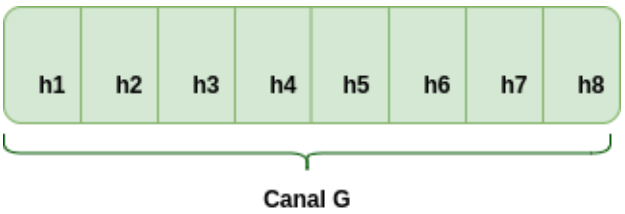
$$b_t[i,j] = \sum_{y=i-br}^{i+br} b_h[y,j]/(2 \cdot br) = \sum_{y=j-br}^{j+br} \left(\sum_{x=j-br}^{j+br} f[y,x]/(2 \cdot br) \right) /(2 \cdot br)$$
$$= \sum_{y=i-br}^{i+br} \sum_{x=j-br}^{j+br} f[y,x]/(2 \cdot br)^2$$

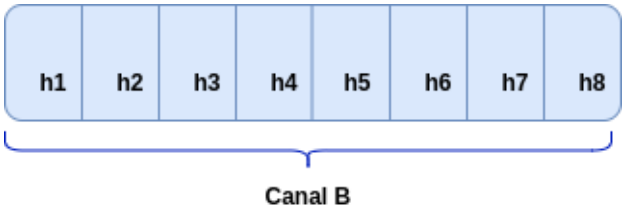
II. Implementacion de la paralelizacion

Con el uso de la librería stb_image.h [3] obtuvimos la representación en los canales RGB de la imagen, luego se aplico el algoritmo por cada canal.

Para lograr la paralelizacion se dividió el canal correspondiente en el numero de hilos y se ejecuto la función de Horizontal blur y Total blur para cada una de estas secciones. En la imagen a continuación se evidencia el particionamiento de la imagen





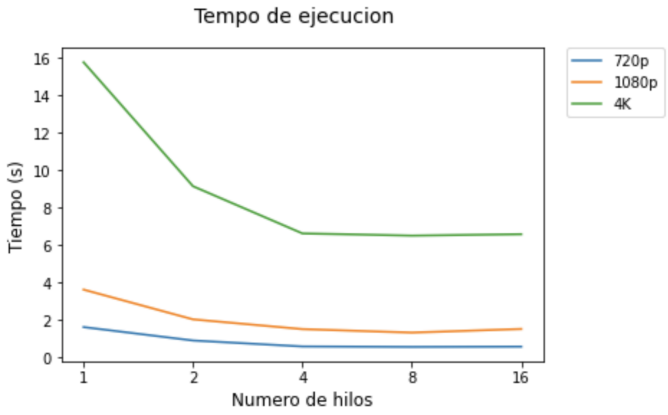


Tal y como se observa se siguió una estrategia **Block-wise** para abordar el problema de la paralelizacion.

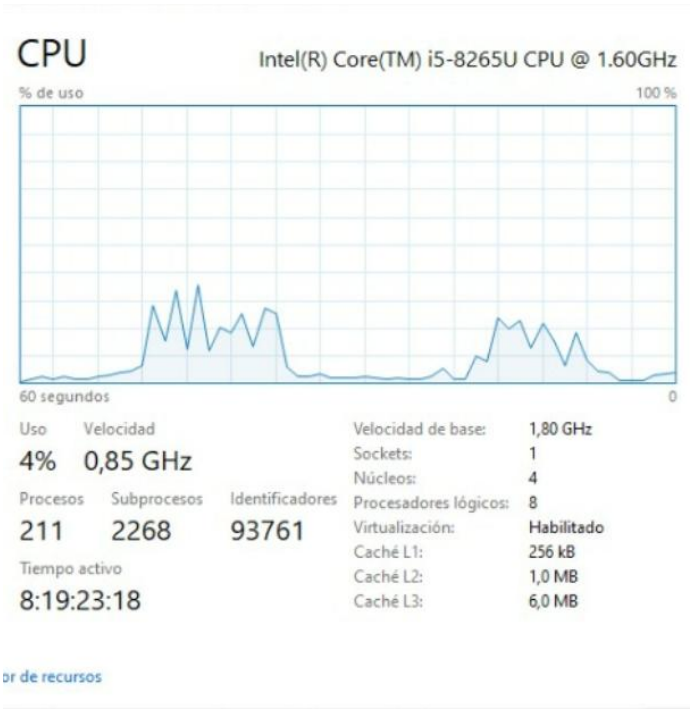
III. Graficas y analisis de resultados

A. Graficas de tiempo de ejecucion

Utilizando un kernel de 9 para el algoritmo de blur construido, se realizaron 10 iteraciones para cada una de las imágenes (720p, 1080p y 4K) y con 1, 2, 4, 8 y 16 hilos. A partir de esto, utilizando el comando de unix "time", se obtuvo el tiempo para cada una de las ejecuciones (50 por imagen) con los parámetros anteriormente mencionados y se saco un promedio de los tiempos obtenidos en cada una de las 10 iteraciones para cada hilo, con esto se formo una gráfica de dichos tiempos de ejecución para cada una de las imágenes. Esta grafica se observa a continuacion.



Como se esperaba de los valores del tiempo entre 1 a 2 hilos, 2 a 4 hilos y 4 a 8 hilos fueron reduciendo considerablemente como producto de la paralelizacion. Sin embargo al pasar de 8 a 16 hilos no se observo una reducción, esto es debido a que la maquina en la que se ejecutó el programa solo cuenta con 4 núcleos físicos (8 núcleos virtuales) por lo que al lanzar 16 hilos quedan virtualizados y no tienen el rendimiento esperado. La especificacion de la maquina se observa a continuacion.

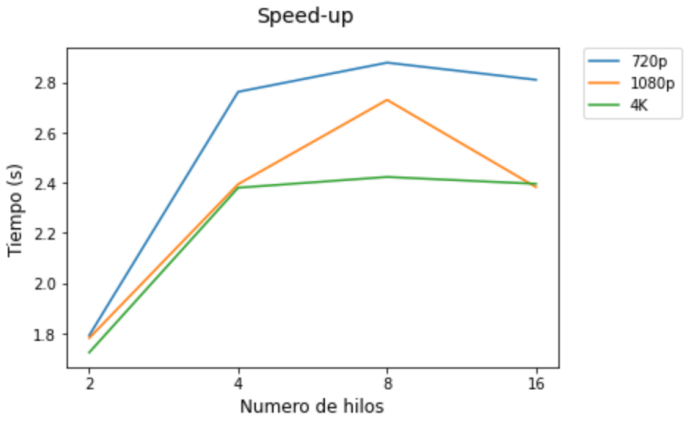


B. Speed-up

Se calculo el speed-up para los tiempos de ejecucion de cada una de las imagenes, esto utilizando la siguiente formula.

$$S_p(n) = \frac{T^*(n)}{T_p(n)}$$

Donde $T*(n)$ es el tiempo del mejor programa secuencial (obtenido a partir de un promedio de 10 ejecuciones con 1 solo hilo) y $T_p(n)$ es el tiempo de ejecución con p hilos. A partir de dichos calculos se obtuvo la siguiente grafica.



Como se esperaba, la grafica cumple con el comportamiento explicado por la ley de Amdhal, los que quiere decir que el

speed up aumenta mientras se acerca a un valor asintotico horizontal. Asimismo, en esta gráfica se observa un dato atípico para el speed up de la imagen de 1080p con 16 hilos, debido a la arquitectura de la maquina donde se ejecuto el programa.

IV. Conclusiones

- Podemos ver como el tiempo de ejecución tiende al mismo comportamiento con los tres tamaños distintos de imagen, esto mostrando un aumento del tiempo de ejecución para el caso de los 16 hilos, sin embargo este aumento no es muy pronunciado.
- Ya que se estaba usando un computador con un procesador de 8 núcleos virtuales, el máximo Speed-up se consigue cuando se ejecuta el programa con 8 hilos ya que luego al ejecutarlo con 16 hilos el Speed-up disminuyo considerablemente para todos los casos, una de las posibles causas de esto es que el Scheduler del computador debió realizar la planificación para la ejecución de los demás hilos. Esto es consecuente al aumento en el caso del tiempo de ejecución.

Bibliografía

[1] Fastest Gaussian Blur (in linear time), [En línea]. Available: http://blog.ivank.net/fastest-gaussian-blur.html [Último acceso: 30 de Marzo 2020].

[2] MATLAB and Octave Functions for Computer Vision and Image Processing, [En línea]. Available: https://www.peterkovesi.com/matlabfns/ [Último acceso: 30 de Marzo 2020]

[3] stb image library C, [En línea]. Available: https://github.com/nothings/stb [Último acceso: 30 de Marzo 2020]

[4] Bash Reference Manual, [En línea]. Available: https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html [Último acceso: 30 de Marzo 2020].