Challenge 2: Histogram of an Image

Diego Madariaga Román

1. Descripción

Este desafío consiste en la implementación en paralelo utilizando la librería *Cilk Plus* de un algoritmo para calcular el histograma de imágenes en formato `PPM` (Portable Pixel Map).

A continuación, se presenta una explicación de la implementación realizada, luego se muestran los resultados experimentales obtenidos y finalmente se realiza un análisis sobre estos resultados, en donde se muestra la escalabilidad de la solución propuesta.

2. Implementación Realizada

La solución propuesta toma como entrada una imagen en formato `PPM` de tamaño $N \times M$, la cual es leída y almacenada en 3 arreglos (`im_R`, `im_G` y `im_B`) de tamaño $N \times M$, cada uno almacenando los valores de `R`, `G` y `B` respectivamente, de cada uno de los píxeles de la imágen. Como ejemplo, el valor de `R` para el píxel de la fila i y columna j se encontraría en `im_R`$[N \times i + j]$.

Una vez leída la imagen en los 3 arreglos mencionados, se calcula el histograma de cada uno de los 3 canales en paralelo. Para cada canal, de forma paralela se calculan histogramas parciales de porciones equitativas de píxeles. Finalmente, se suman secuencialmente los histogramas parciales para construir el histograma final del canal.

Para el análisis teórico consideramos solamente el cálculo del histograma en sí y no el proceso de lectura de la imagen mencionado anteriormente, ya que esta parte no ha de ser paralelizada. En el cálculo del histograma para un canal (se calcula solo para uno ya que se ejecutan en paralelo) se tiene que $T_P = O\left(log(P) \times \frac{N \times M}{P}\right)$ ya que en el caso de tener P procesadores, cada uno de ellos calcula en paralelo el histograma parcial de una porción de $\frac{N \times M}{P}$ píxeles. Luego, para sumar los P histogramas parciales generados, se deben recorrer secuencialmente sumando sus 256 valores, en donde se tiene que $T_P = O(P)$.

Dado lo anterior, se tiene que:

$$T_P = O\left(log(P) \times \frac{N \times M}{P}\right) + O(P)$$

$$T_P = O\left(log(P) \times \frac{N \times M}{P} + P\right)$$

por lo que la cantidad óptima de procesadores es donde se minimice $\{log(P) \times \frac{N \times M}{P} + P\}$.

El trabajo T_1 que se obtiene es $O(N \times M)$ y corresponde a calcular el histograma leyendo píxel por píxel y canal por canal. Luego, para P procesadores, el máximo paralelismo esperado está dado por:

$$\frac{O\left(log(P) \times \frac{N \times M}{P} + P\right)}{O(N \times M)}$$

3. Experimentación

El código utilizado y las instrucciones para reproducir los experimentos realizados se encuentran disponibles en `https://git.io/dmadariaga-histogram`

Para la experimentación realizada se utilizaron 3 imágenes de prueba creadas con el archivo `random_rgb_image.c`:

- `2048.ppm`: Imagen de tamaño $2048 \times 2048 = 4194304$ píxeles.
- `4096.ppm`: Imagen de tamaño $4096 \times 4096 = 16777216$ píxeles.
- `8192.ppm`: Imagen de tamaño $8192 \times 8192 = 67108864$ píxeles.

Con cada imagen, se ejecutó el programa paralelo `histogram_par.c` implementado, utilizando desde 1 a 12 procesadores y midiendo el tiempo de ejecución y el número de *cache-misses*.

Cada experimento fue realizado 3 veces y se han reportado los valores correspondientes a las medianas obtenidas en cada caso.

Para los cálculos de *speedup* se utilizó como tiempo secuencial el tiempo registrado al ejecutar el archivo `histogram_par.c` en su versión `NOPARALLEL`, en la cual se eliminan las ocurrencias del *keyword* `cilk_spawn` y se cambia la ocurrencia de `cilk_for` por su versión secuencial `for`.

Todos los experimentos fueron llevados a cabo utilizando un computador de 12 cores físicos, distribuidos en una arquitectura NUMA de 2 nodos, cada uno con 6 cores y 3GB de memoria RAM. Los resultados se muestran a continuación:

Figura 3.1: Tiempo en [s] para el cálculo de histograma de imagen de tamaño $2048 \times 2048 = 4194304$ píxeles.

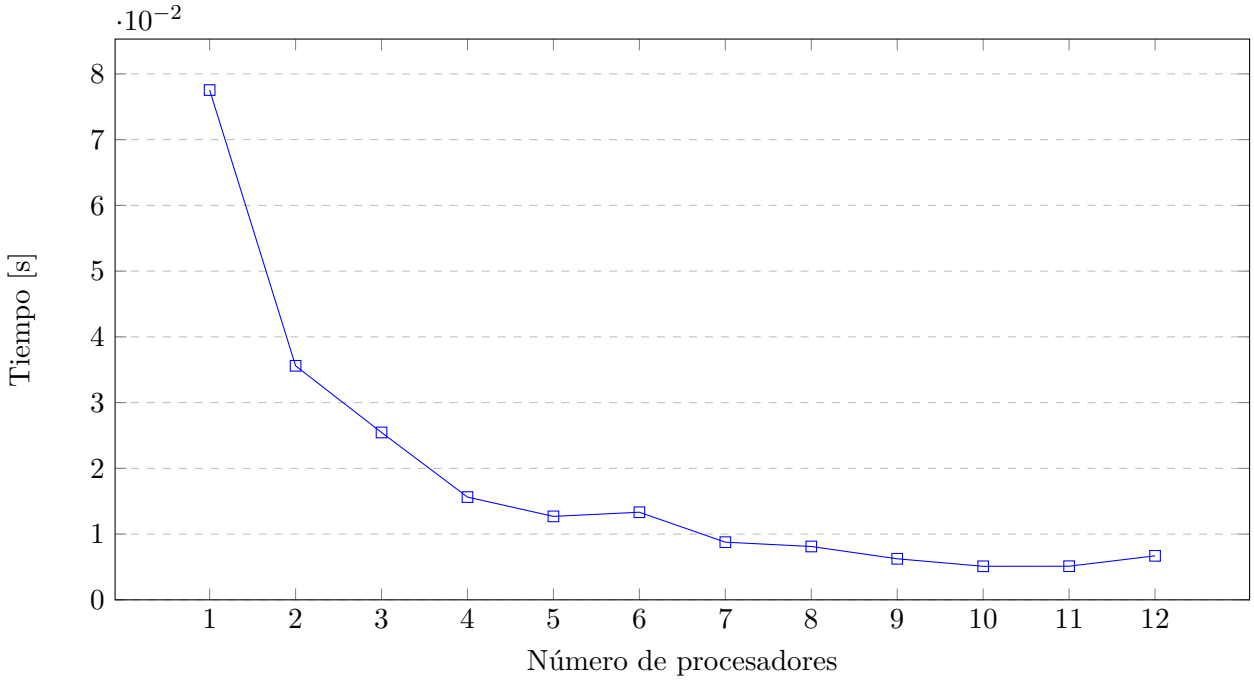


Figura 3.2: Tiempo en [s] para el cálculo de histograma de imagen de tamaño $4096 \times 4096 = 16777216$ píxeles.

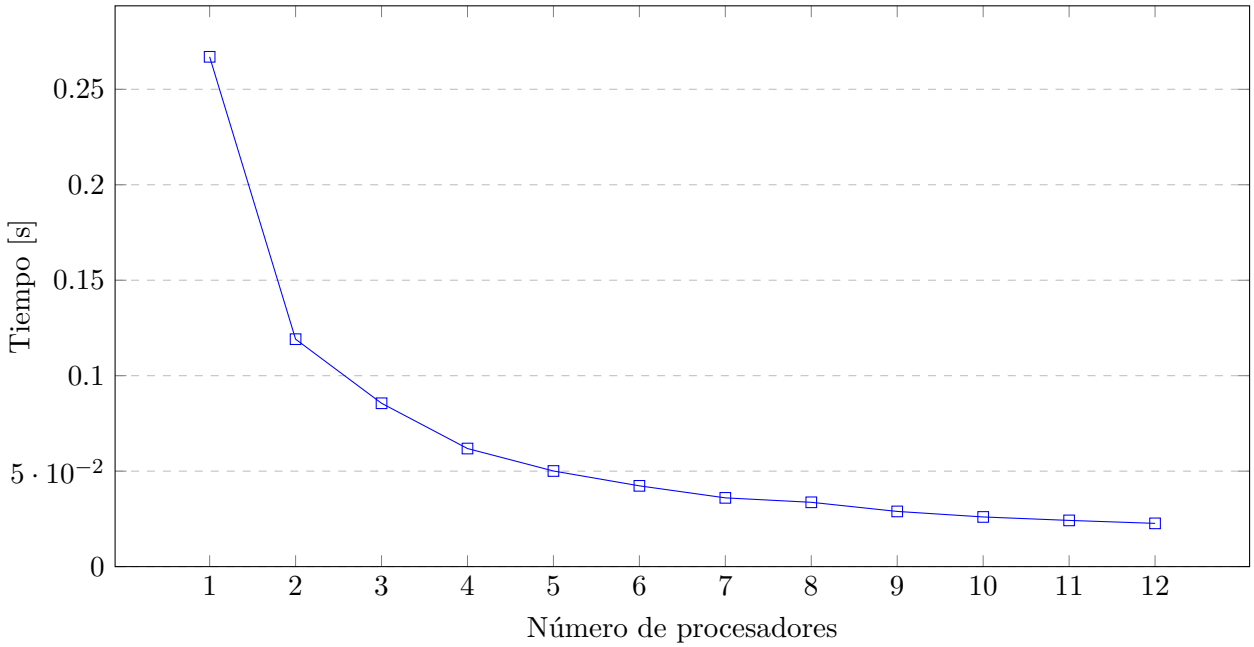


Figura 3.3: Tiempo en [s] para el cálculo de histograma de imagen de tamaño $8192 \times 8192 = 67108864$ píxeles.

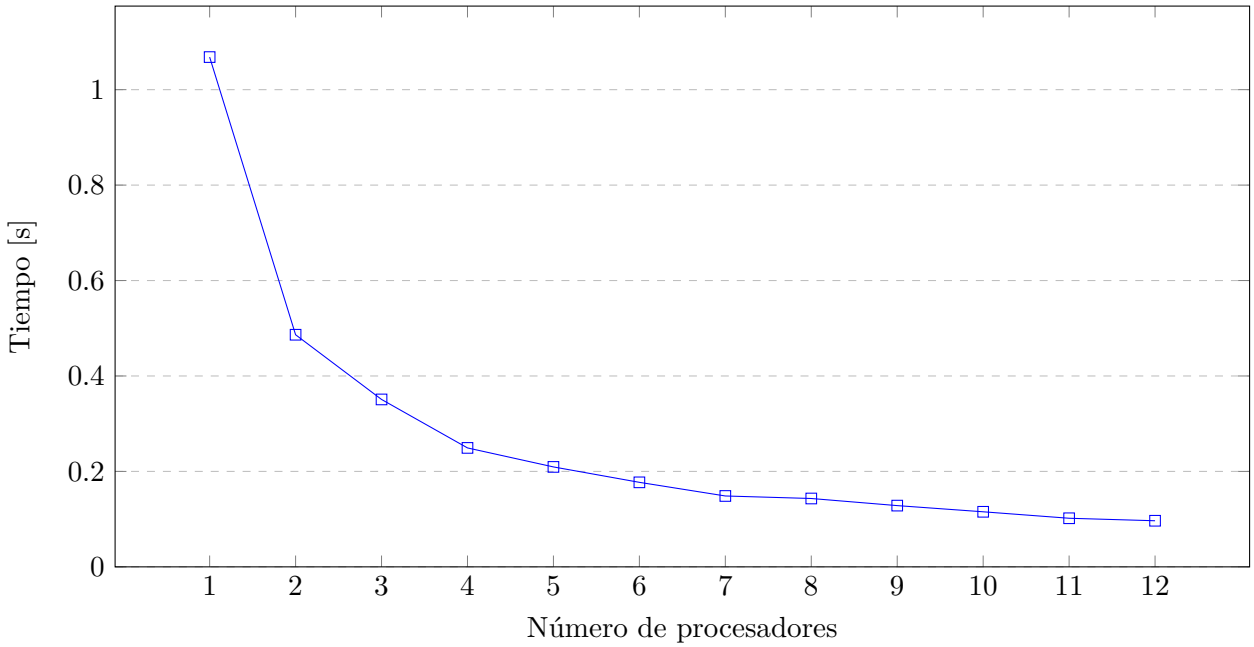


Figura 3.4: *Speedup* obtenido en el cálculo de histograma para distintos tamaños de imágenes.

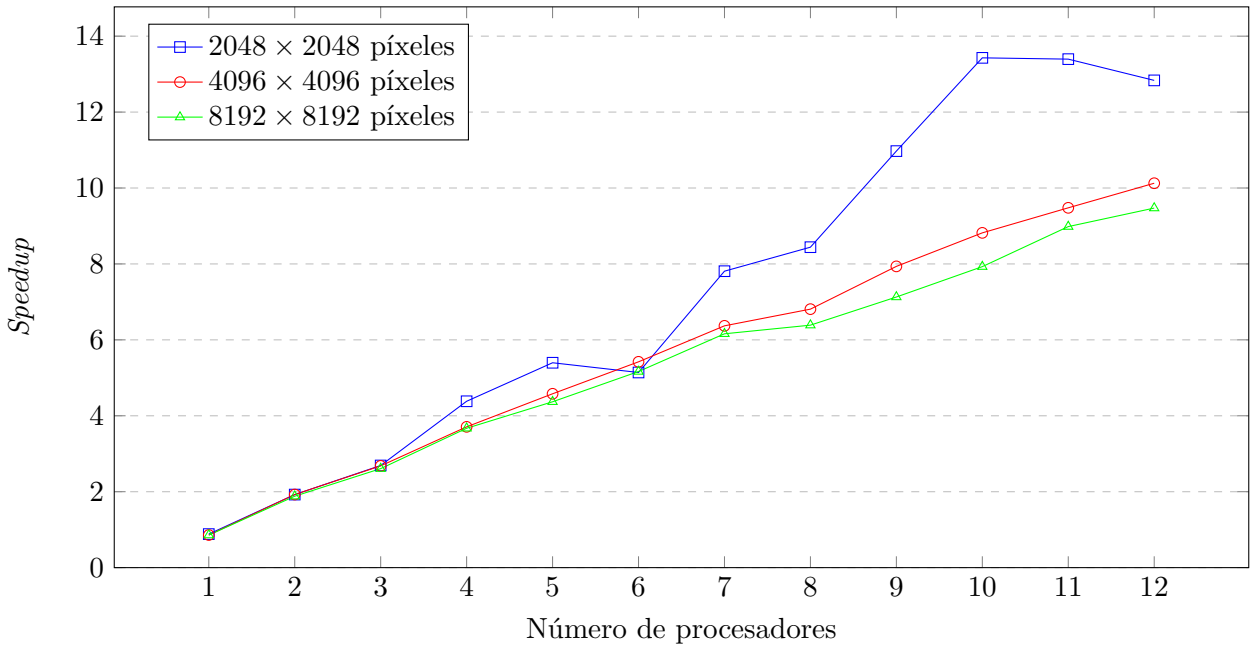
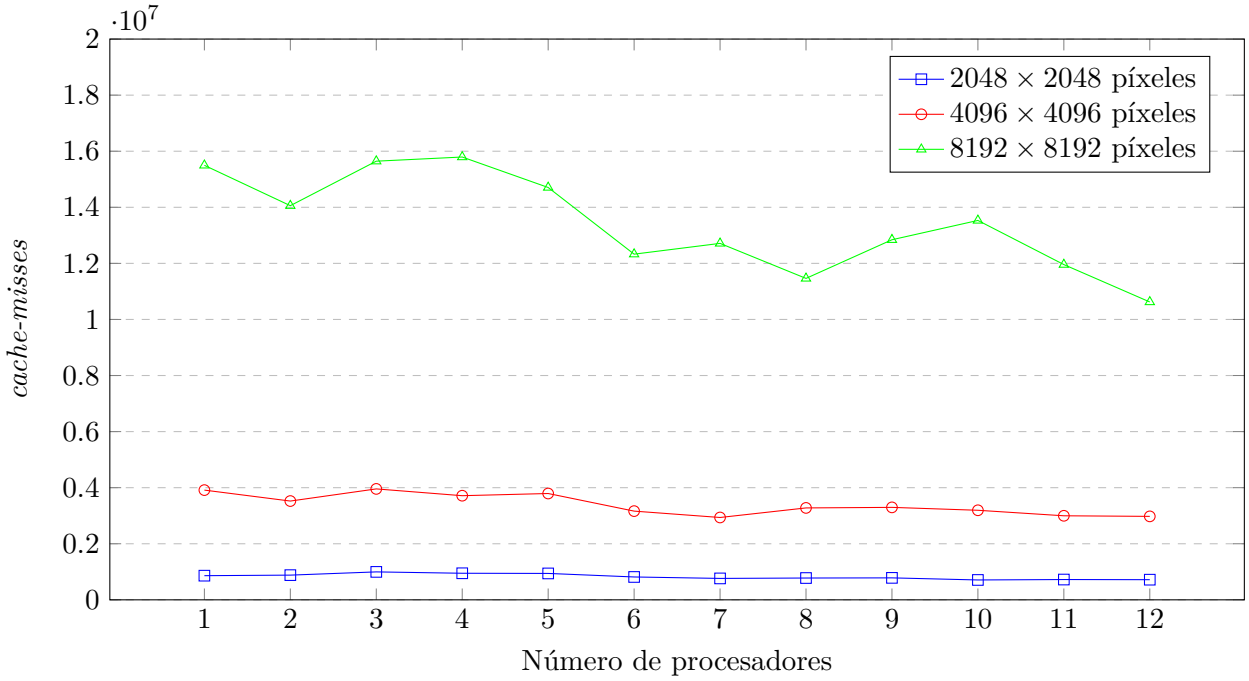


Figura 3.5: Número de *cache-misses* durante el cálculo de histograma para distintos tamaños de imágenes.



4. Discusión y Conclusiones

En las Figuras 3.1, 3.2 y 3.3 se puede observar que para las 3 imágenes de prueba, se disminuye el tiempo de ejecución al aumentar el número de *cores* utilizados. Dichas imágenes poseen curvas similares, en las que se encuentran las primeras indicaciones de la buena escalabilidad del algoritmo.

La Figura 3.4, que muestra los valores de *speedup* obtenidos utilizando las 3 imágenes de prueba, refleja en las curvas de las imágenes de 4096×4096 y 8192×8192 píxeles que para el uso de los 12 procesadores, se obtienen valores cercanos a 10, lo cual es muy bueno y da cuentas de la buena escalabilidad del algoritmo. Es importante notar que aunque la curva que representa a la imagen de 2048×2048 píxeles pareciese ser auspiciosa dado que para el uso de más de 6 procesadores se obtienen valores de *speedup* super lineales, estos resultados no han sido considerados, ya que se estima que dichos valores no representan el comportamiento real del algoritmo. Esto se explica dado que todos los experimentos realizados con esta imagen entregaron tiempos de ejecución muy bajos, por lo que pequeñas variaciones ocurridas en el computador durante dichos experimentos se reflejan como importantes anomalías en los resultados.

Finalmente, la Figura 3.5 muestra que el número de *cache-misses* no depende del número de procesadores utilizados, sino que del tamaño de la imagen procesada.