

# Computación Paralela y Distribuida

## Practica # 2

José Fernando Morantes Florez, Johan Sebastian 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 más 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.

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

Como podemos observar esta fórmula se aplica a cada píxel 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 más eficiente se dividió la operación de convolución en 2, en la primera de estas (Horizontal Blur) se itera sobre las columnas dejando estática la fila y se aplica la siguiente formula.

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

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

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

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

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

### II. IMPLEMENTACIÓN DE LA PARALELIZACIÓN

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

Para lograr la paralelización se dividió el canal correspondiente en el número de hilos y se ejecutó 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.

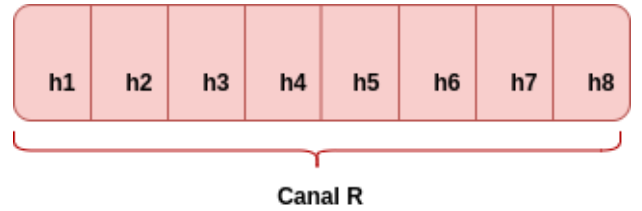


Fig. 1. Partición del canal rojo

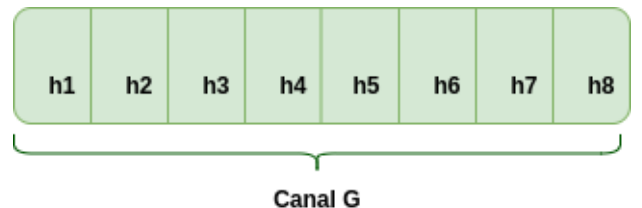


Fig. 2. Partición del canal verde

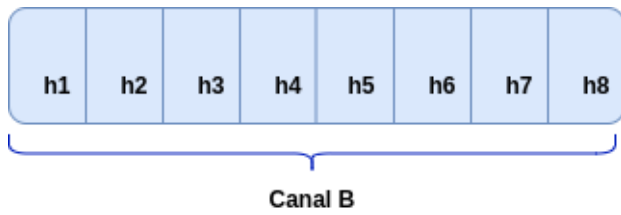


Fig. 3. Partición del canal azul

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

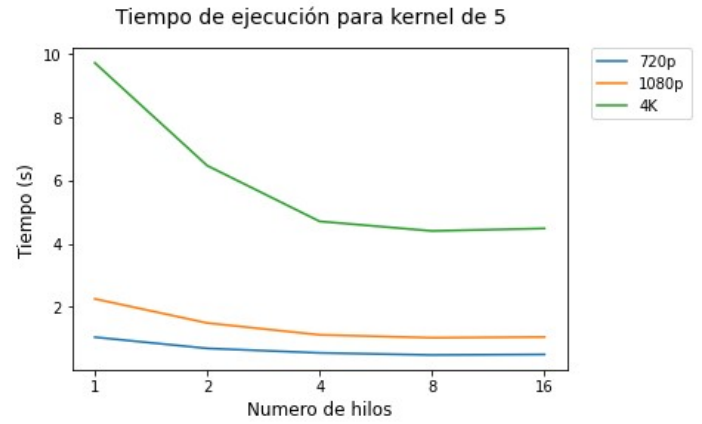


Fig. 5. Grafica de tiempo de ejecución para las 3 imágenes con un kernel de 5 utilizando POSIX

### III. GRÁFICAS Y ANÁLISIS DE RESULTADOS UTILIZANDO POSIX

#### A. Graficas de tiempo de ejecución

Utilizando un kernel de 3, 5, 7, 9, 11, 13 y 15 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 sacó un promedio de los tiempos obtenidos en cada una de las 10 iteraciones para cada hilo y cada kernel, con esto se formó una gráfica de dichos tiempos de ejecución para cada uno de los kernel con las tres imágenes. Estas se muestran a continuación.

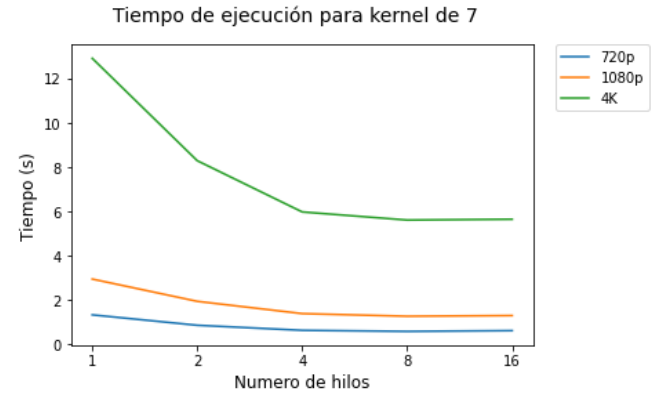


Fig. 6. Grafica de tiempo de ejecución para las 3 imágenes con un kernel de 7 utilizando POSIX

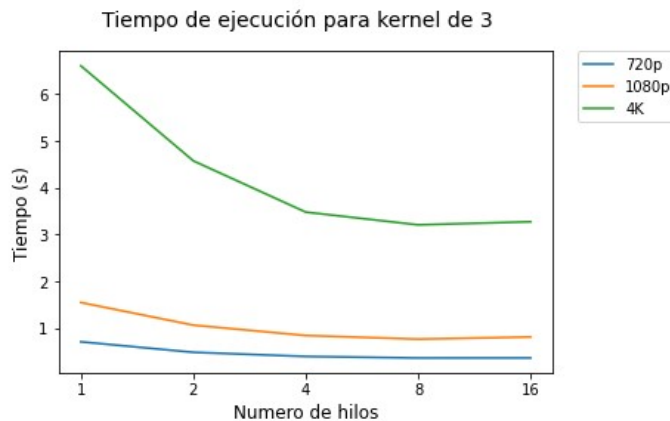


Fig. 4. Grafica de tiempo de ejecución para las 3 imágenes con un kernel de 3 utilizando POSIX

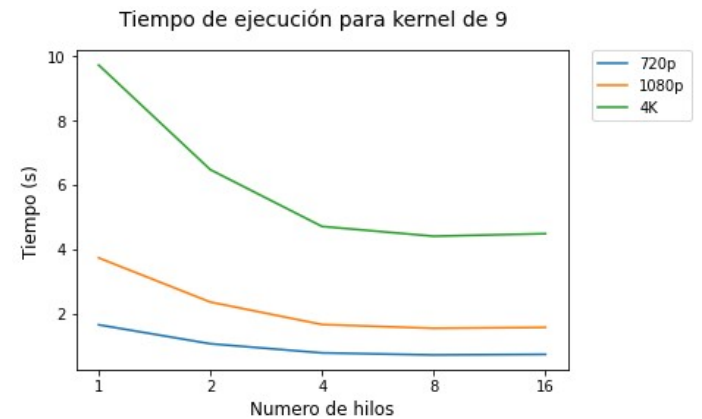


Fig. 7. Grafica de tiempo de ejecución para las 3 imágenes con un kernel de 9 utilizando POSIX

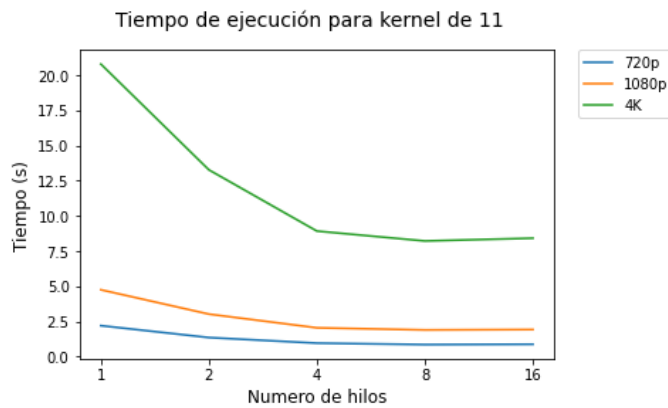


Fig. 8. Gráfica de tiempo de ejecución para las 3 imágenes con un kernel de 11 utilizando POSIX

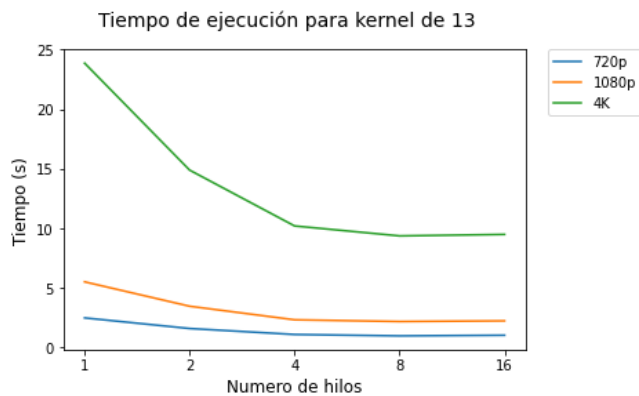


Fig. 9. Gráfica de tiempo de ejecución para las 3 imágenes con un kernel de 13 utilizando POSIX

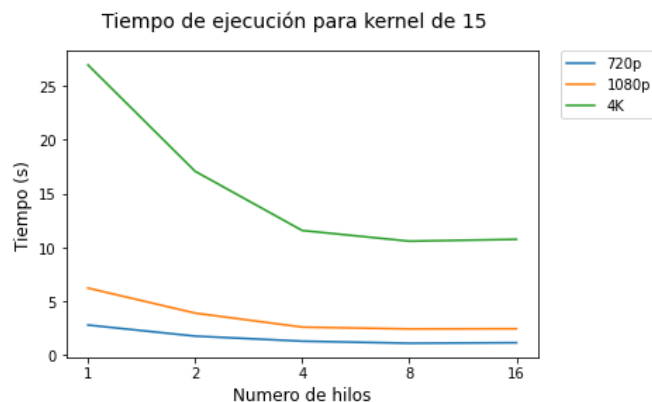


Fig. 10. Gráfica de tiempo de ejecución para las 3 imágenes con un kernel de 15 utilizando POSIX

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 paralelización en todos los casos. Sin embargo al pasar de 8 a 16 hilos no se observó una reducción

en su lugar se mantuvo constante o se notó un ligero aumento en el tiempo de ejecución, esto es debido a que la maquina en la que se ejecutó el programa solo cuenta con 4 núcleos físicos y por lo tanto 8 núcleos virtuales esto implica que al lanzar 16 hilos quedan virtualizados y no tienen el rendimiento esperado.

Cabe resaltar que el aumento en el tamaño del kernel representa para el algoritmo un aumento en el tiempo de ejecución, este aumento es proporcional en cada uno de los tamaños de kernel por lo que las gráficas siguen la misma tendencia.

La especificación de la maquina se observa a continuación.

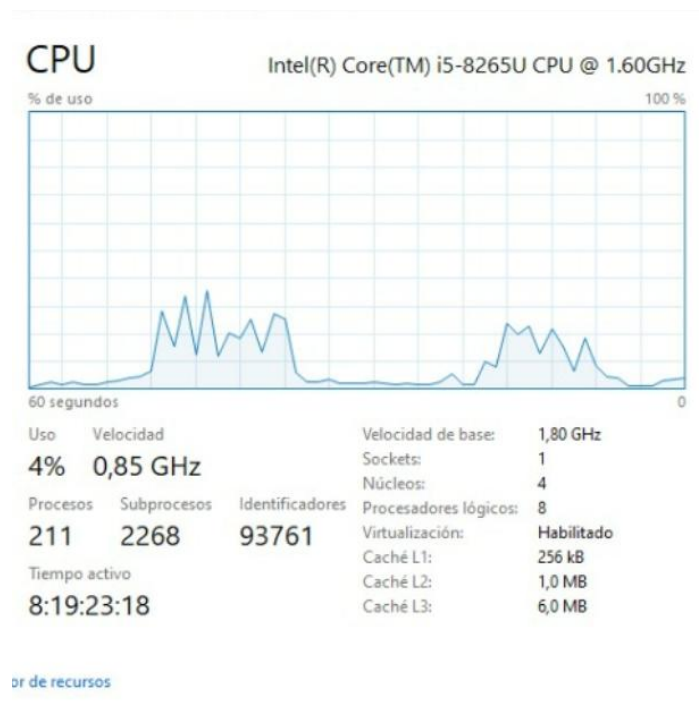


Fig. 11. Detalles de la maquina donde se ejecuto el algoritmo

### B. Speed-up

Se calculo el speed-up para los tiempos de ejecución anteriormente obtenidos, 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 cálculos se obtuvo la siguiente grafica.

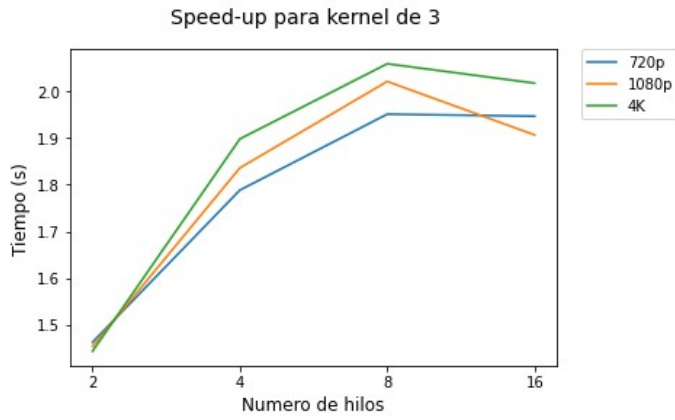


Fig. 12. Grafica del speed-up para las 3 imágenes con un kernel de 3 utilizando POSIX

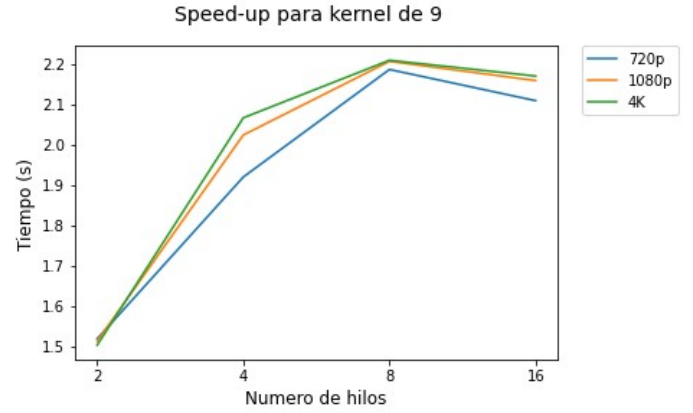


Fig. 15. Grafica del speed-up para las 3 imágenes con un kernel de 9 utilizando POSIX

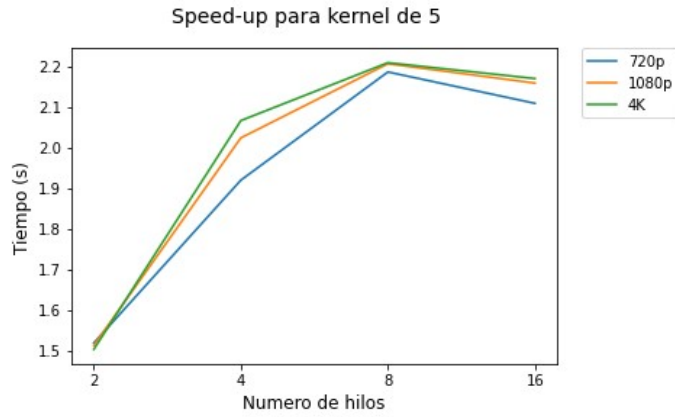


Fig. 13. Grafica del speed-up para las 3 imágenes con un kernel de 5 utilizando POSIX

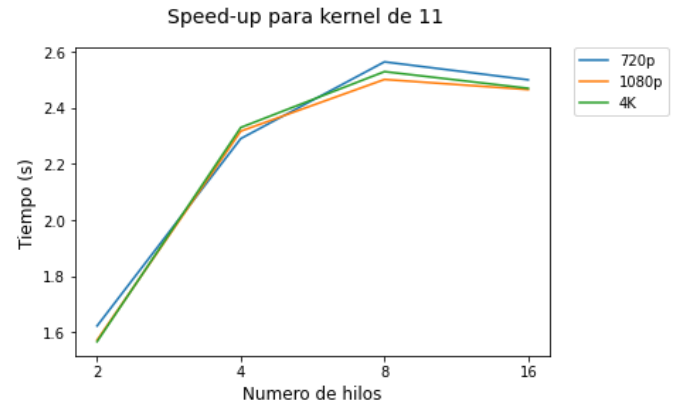


Fig. 16. Grafica del speed-up para las 3 imágenes con un kernel de 11 utilizando POSIX

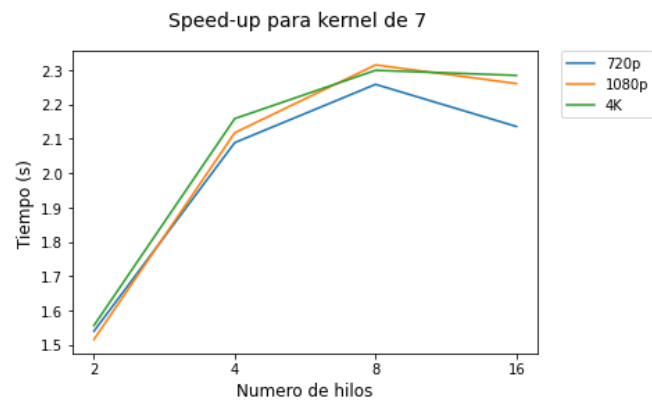


Fig. 14. Grafica del speed-up para las 3 imágenes con un kernel de 7 utilizando POSIX

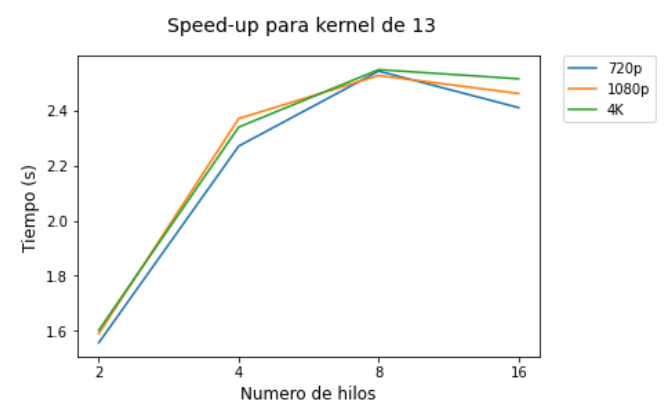


Fig. 17. Grafica del speed-up para las 3 imágenes con un kernel de 13 utilizando POSIX

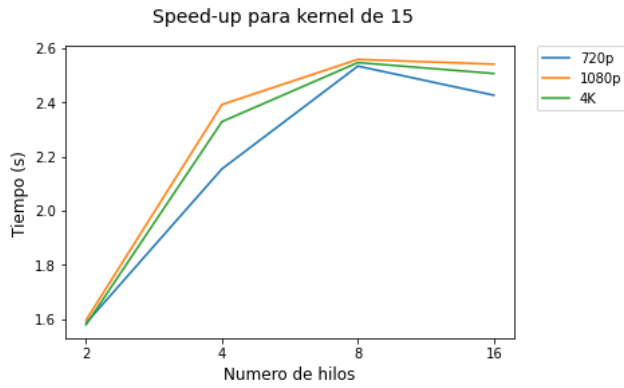


Fig. 18. Gráfica del speed-up para las 3 imágenes con un kernel de 15 utilizando POSIX

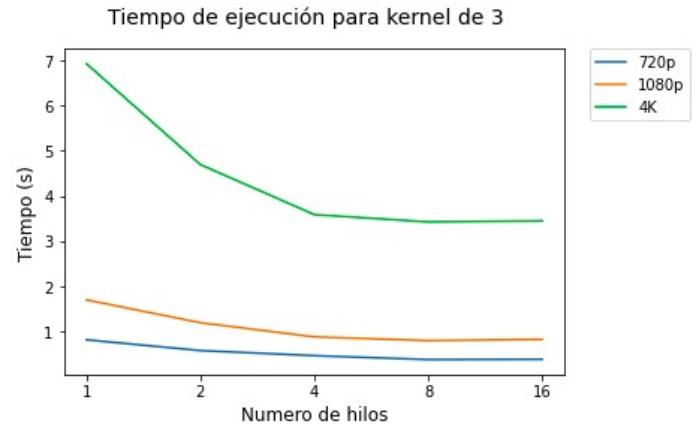


Fig. 19. Gráfica de tiempo de ejecución para las 3 imágenes con kernel de 3 utilizando OpenMP

Como se esperaba, las gráficas cumplen con el comportamiento explicado por la ley de Amdhal, los que quiere decir que el speed up aumenta mientras se acerca a un valor asintótico horizontal. También podemos ver que en casos particulares como el observado en la Fig.12 (Kernel 3) se observan datos atípicos en el proceso usando 16 hilos, específicamente en el caso de la imagen a 1080p, esto puede ser debido a la arquitectura de la maquina donde se ejecutó el programa.

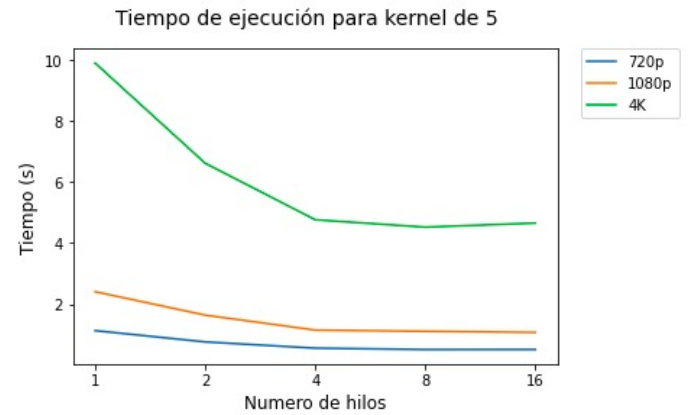


Fig. 20. Gráfica de tiempo de ejecución para las 3 imágenes con kernel de 5 utilizando OpenMP

#### IV. GRÁFICAS Y ANÁLISIS DE RESULTADOS UTILIZANDO OPENMP

##### A. Gráficas de tiempo de ejecución

Utilizando un kernel de 3, 5, 7, 9, 11, 13 y 15 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 sacó un promedio de los tiempos obtenidos en cada una de las 10 iteraciones para cada hilo y cada kernel, con esto se formó una gráfica de dichos tiempos de ejecución para cada uno de los kernel con las tres imágenes. Estas se muestran a continuación.

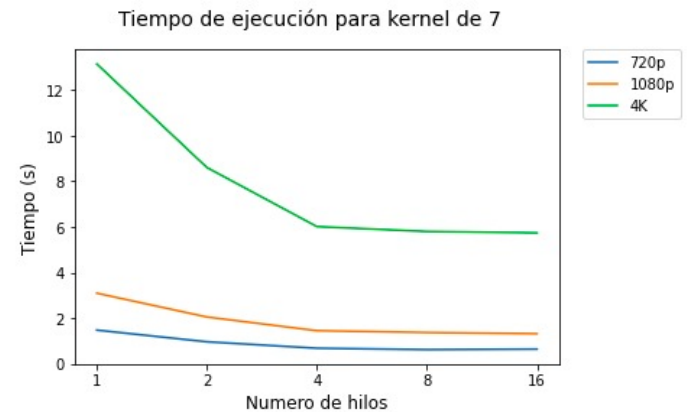


Fig. 21. Gráfica de tiempo de ejecución para las 3 imágenes con kernel de 7 utilizando OpenMP

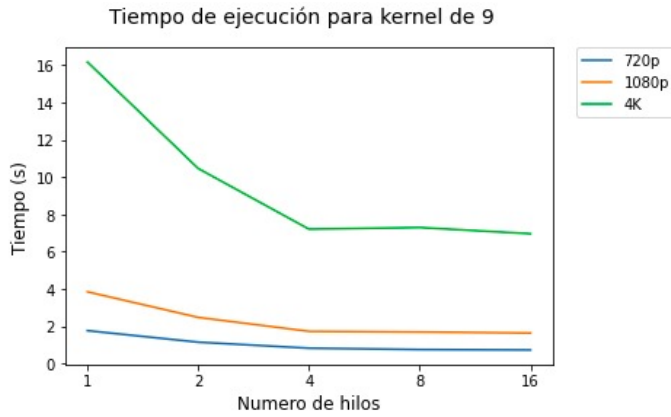


Fig. 22. Gráfica de tiempo de ejecución para las 3 imágenes con un kernel de 9 utilizando OpenMP

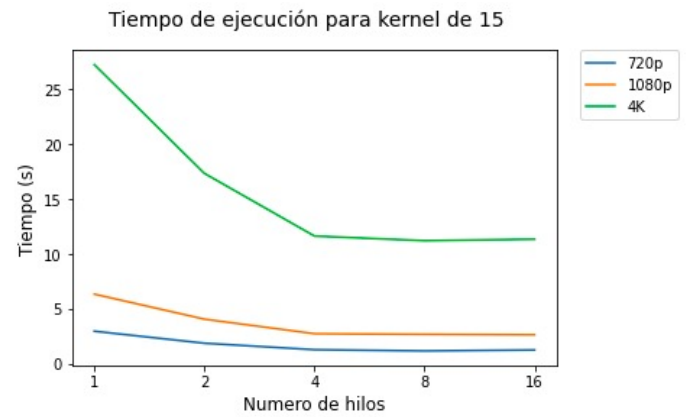


Fig. 25. Gráfica de tiempo de ejecución para las 3 imágenes con un kernel de 15 utilizando OpenMP

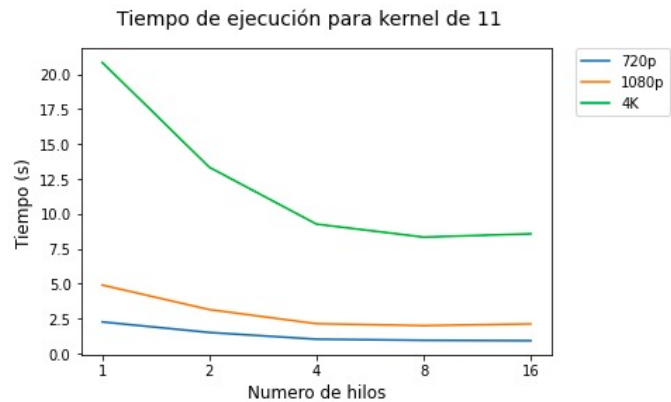


Fig. 23. Gráfica de tiempo de ejecución para las 3 imágenes con un kernel de 11 utilizando OpenMP

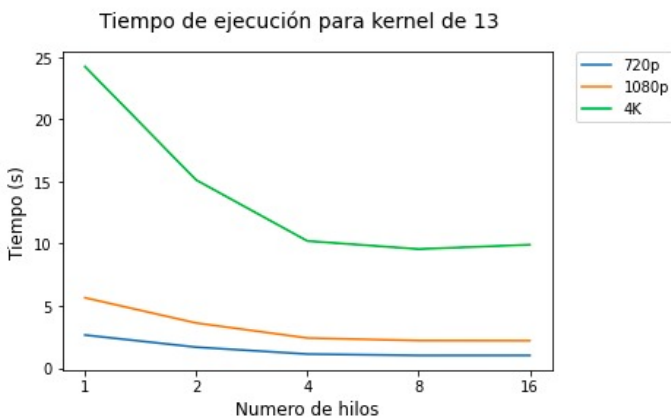


Fig. 24. Gráfica de tiempo de ejecución para las 3 imágenes con un kernel de 13 utilizando OpenMP

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 paralelización en todos los casos. Sin embargo al pasar de 8 a 16 hilos, al igual que con la implementación utilizando POSIX, no se observó una reducción en su lugar se mantuvo constante o se notó un ligero aumento en el tiempo de ejecución, esto es debido a que la maquina en la que se ejecutó el programa solo cuenta con 4 núcleos físicos y por lo tanto 8 núcleos virtuales esto implica que al lanzar 16 hilos quedan virtualizados y no tienen el rendimiento esperado.

### B. Speed-up

Se calculo el speed-up para los tiempos de ejecución anteriormente obtenidos.

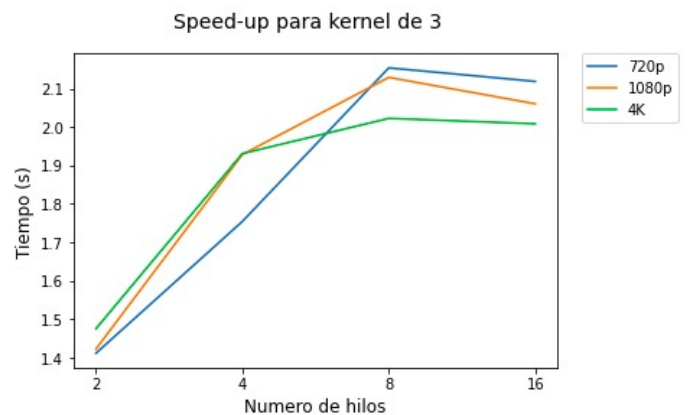


Fig. 26. Gráfica del speed-up para las 3 imágenes con un kernel de 3 utilizando OpenMP

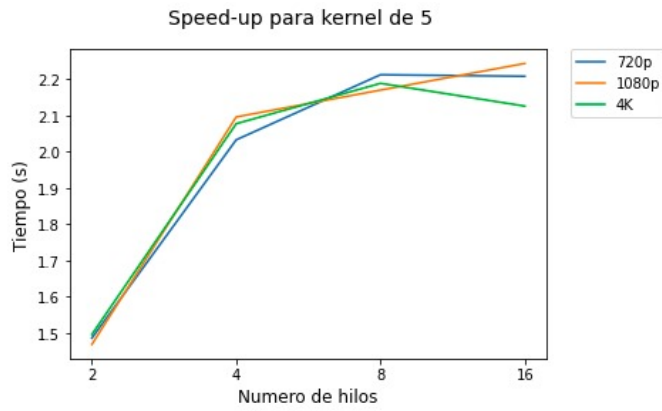


Fig. 27. Gráfica del speed-up para las 3 imágenes con un kernel de 5 utilizando OpenMP

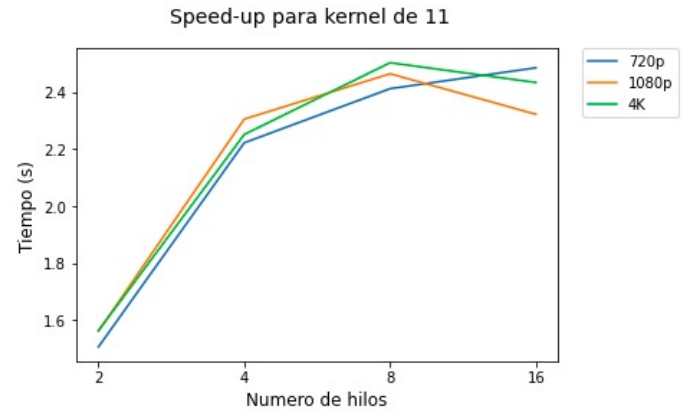


Fig. 30. Gráfica del speed-up para las 3 imágenes con un kernel de 11 utilizando OpenMP

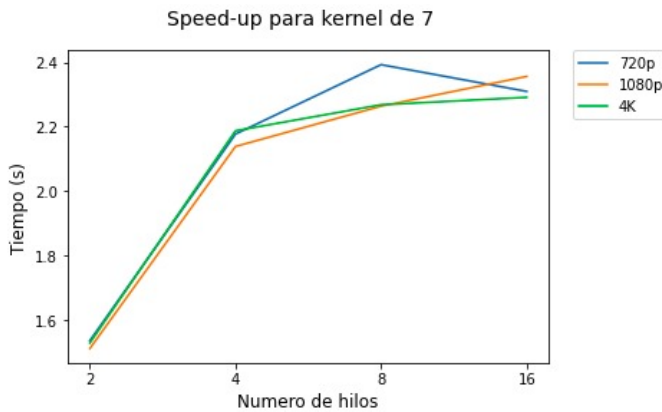


Fig. 28. Gráfica del speed-up para las 3 imágenes con un kernel de 7 utilizando OpenMP

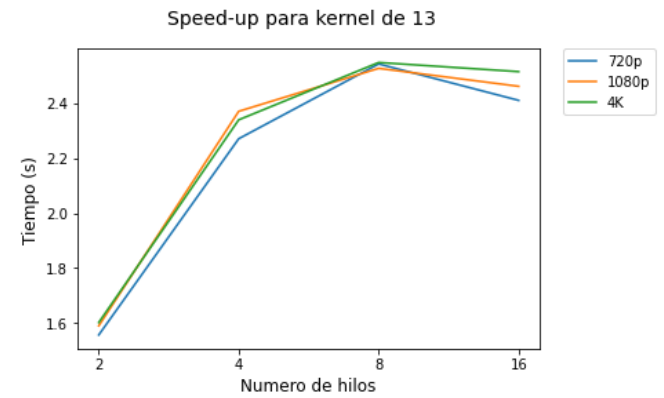


Fig. 31. Gráfica del speed-up para las 3 imágenes con un kernel de 13 utilizando OpenMP

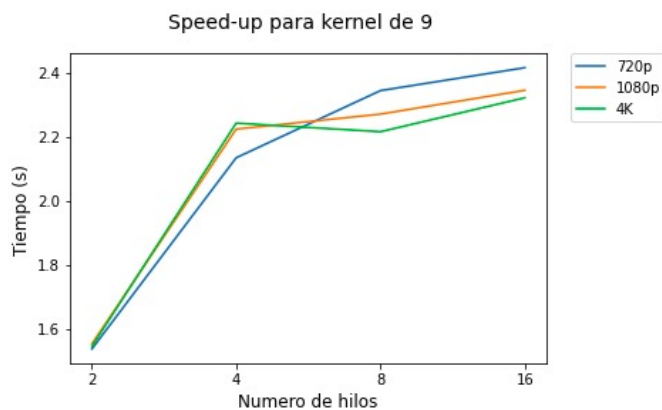


Fig. 29. Gráfica del speed-up para las 3 imágenes con un kernel de 9 utilizando OpenMP

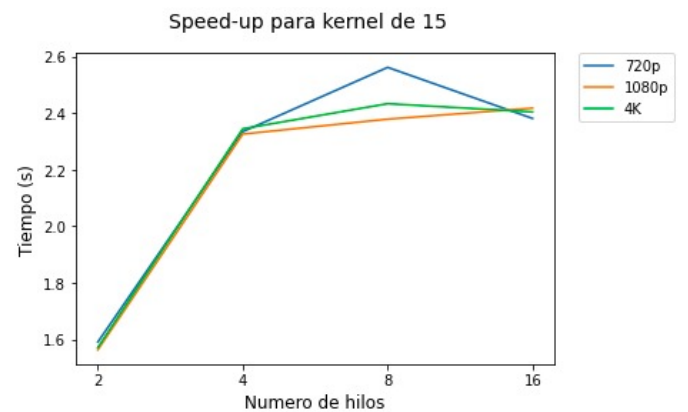


Fig. 32. Gráfica del speed-up para las 3 imágenes con un kernel de 15 utilizando OpenMP

Como se esperaba, las gráficas cumplen con el comportamiento explicado por la ley de Amdahl, los que quiere decir que el speed up aumenta mientras se acerca a un valor asintótico horizontal. También podemos ver que en



casos particulares como el observado en la Fig.12 (Kernel 3) se observan datos atípicos en el proceso usando 16 hilos, específicamente en el caso de la imagen a 1080p, esto puede ser debido a la arquitectura de la maquina donde se ejecutó el programa.

## V. COMPARACIÓN ENTRE LOS METODOS

Con respecto al tiempo de ejecución se observó especialmente en los kernel bajos que la implementación con OpenMP posee tiempos de ejecución mayores con respecto a la implementación en POSIX usando los mismos parámetros, esto puede ser debido a que OpenMP compensa la facilidad y simplicidad a lo hora de escribir código con el rendimiento de este, cabe aclarar que esta diferencia de tiempo es bastante pequeña, aproximadamente del orden de un segundo. Por otra parte, en los kernel más altos podemos ver que el comportamiento del tiempo de ejecución entre ambas implementaciones es bastante similar, no se observan cambios bruscos ni comportamientos anormales al contrastar los datos.

Un comportamiento atípico se puede observar en el tiempo de ejecución de la implementación de OpenMP para un kernel de 9, donde podemos ver que el tiempo de ejecución con 1 hilo es inusualmente alto, esto puede ocurrir por muchas razones, la más probable es un cambio en las condiciones de la máquina al momento de su ejecución, por ejemplo un proceso en segundo plano disparado en ese preciso instante.

Con respecto al Speed-up y teniendo en cuenta el análisis de tiempo de ejecución ya explicado, se puede ver que en para la mayoría de los kernel (con una excepción clara del kernel 9 por su comportamiento atípico) el speed-up al aplicar el algoritmo en imágenes de 4k es menor, esto contrastado con estas mismas ejecuciones en la implementación de POSIX.

## VI. CONCLUSIONES

- Podemos ver como el tiempo de ejecución tiende al mismo comportamiento con los tres tamaños distintos de imagen sin importar el kernel, esto mostrando un aumento del tiempo de ejecución para el caso de los 16 hilos, sin embargo, este aumento no es muy pronunciado, esto aplica tanto para la implementación con OpenMP como para la implementación con POSIX.
- 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 disminuyó 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. NO hubo variación del comportamiento en este caso al cambiar la implementación de POSIX a OpenMP por lo que esto es independiente al método de paralelización y solo depende de las características de la máquina en que es ejecutado.
- Se observa que para todos los tamaños de kernel y tamaños de imagen, el algoritmo mantiene su comportamiento y este es por lo tanto independiente de la

cantidad de hilos que se lance siempre y cuando no se sobrepase la cantidad de núcleos virtuales lo cual como ya se vio causa un aumento en los tiempos de ejecución.

- A nivel general se observó que a pesar de que OpenMP presenta una mayor facilidad en cuanto a su implementación en el código fuente, esto no supone una mejora en el rendimiento de la ejecución, por el caso contrario, en algunos casos vemos que los tiempos de ejecución son mayores con respecto a la implementación en POSIX.

## 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].