Práctica 2: Efecto difuminado usando OpenMP

Andrés Fernando Román Arévalo, Ronald Alexander Sarmiento
Email:afromana@unal.edu.co, roasarmientoga@unal.edu.co
Universidad Nacional Bogotá, Colombia

***Resumen*—El siguiente informe presenta como se llevo a cabo la paralelización mediante el uso de la interfaz de programación openMP del efecto borroso y los resultados que se obtuvieron tanto de Speed Up como de Time Response para los distintas resoluciones de imagenes (720p, 1080p y 4k), hilos(1, 2, 4, 8 y 16) y kernels (4, 6, 8, 10, 12 y 14).**

I. Introducción.

Una de las tareas altamente demandada en la actualidad es el procesamiento de imagenes cada vez de una mayor resolución para lo cual se requiere un alto nivel de computo. A partir de esto han surgido innumerables algoritmos que permiten paralelizar las distintas operaciones del procesamiento de imagenes como lo es la del efecto difuminado.

Aun asi paralelizar tiene su limite que es explicado en la Ley de Amdahl, la cual establece que hay un punto en que paralelizar ya no ofrece mejoras significativas de rendimiento e incluso por el contrario puede llevar al deterioro del mismo.

Al momento de hacer procesamiento de imagenes se utiliza distintas operaciones matriciales.En el caso del efecto borroso se realiza una convolucion entre la matriz que representa la imagen y el kernel.

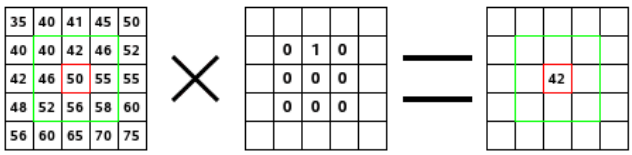


Figura 1: Convolucion matricial

I-A. Creacion del efecto difuminado

Se utilizó la tecnica de difuminado conocida como Blur el cual consiste en intercambiar cada pixel de la imagen por un promedio de los valores RGB que se encuentras en los pixeles adyacentes a el. El nivel de difuminado esta determinado por el kernel y el tamaño da el rango que se toma para evaluar el promedio. Es decir, si se escoge un kernel de 5 significa que cada pixel sera cambiado por el prmedio que hay en los promedios de la matriz 5x5 cuyo centro es el pixel en cuestion.

II. Paralelización del algoritmo

La paralelizacion se realizo mediante asignación tipo block-wise es decir que se tomo la imagen y se dividio a lo ancho en columnas y cada una de estas se asigno a un hilo. Asi, para un caso hipotetico donde se quiera aplicar el efecto difuminado a una imagen con un ancho de 700 pixeles de ancho,se lanzan 4 hilos donde el hilo 1 se le asigna la columna 1(del pixel 0 al 175), al 2 la columna 2(del pixel 176 al 350), al 3 la columna 3(del pixel 351 al 525) y al 4 la columna 4(del pixel 326 al 700). De esta manera, cada hilo tiene para procesar una imagen de $(width/THREADS)x\ height$ como se muestra en la Figura **??**.

Figura 2: Algoritmo de paralelización

III. Experimentos y resultados.

Antes de mostrar los resultados obtenidos y compararlos con los que se obtuvieron al paralelizar el efecto difuso con POSIX, es necesario dar a conocer las especificaciones correspondientes del ordenador donde corrio el algoritmo. Para esto se construyo la siguiente tabla resumiendo dicha información:

Especificaciones	
Modelo	Lenovo G50-45
Procesador	AMD A8-6410 / 2 Gz
Tarjeta gráfica	AMD Radeon R5
Nucleos	4
Memoria Ram	8192 MB DDR3
Sistema operativo	Manjaro 18.0.4

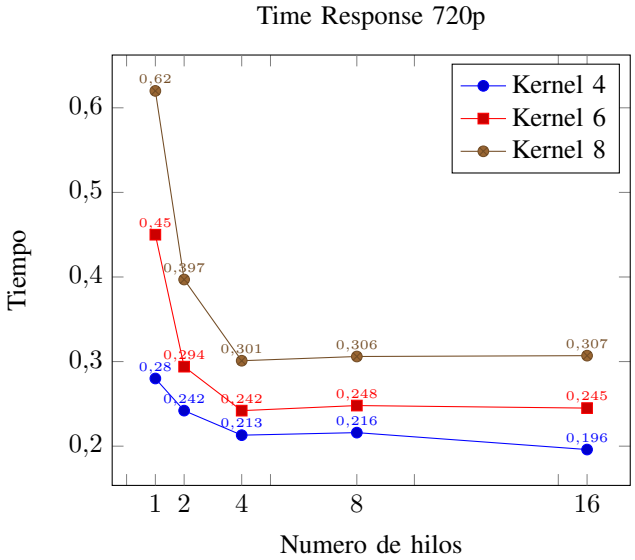
Cuadro I: Especificaciones del ordenador

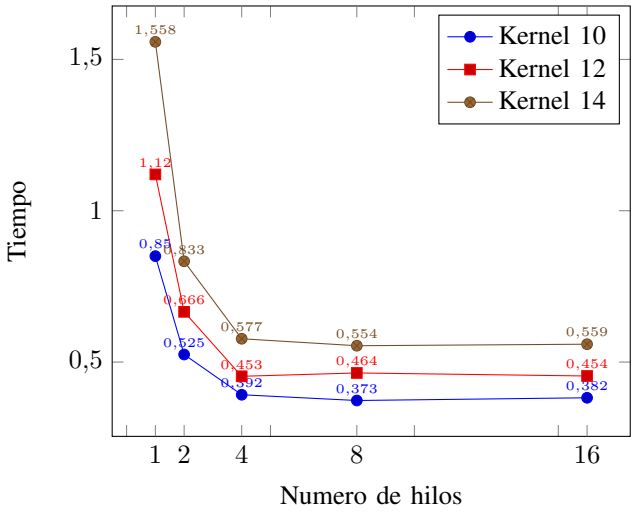
Para evitar o disminuir posibles interferencias se corrió un script que automaticamente tomó todos los casos de prueba mencionados en el resumen, tambien se busco que no hubiese ninguna aplicacion abierta mientras se corria el script.

III-A. Resultados

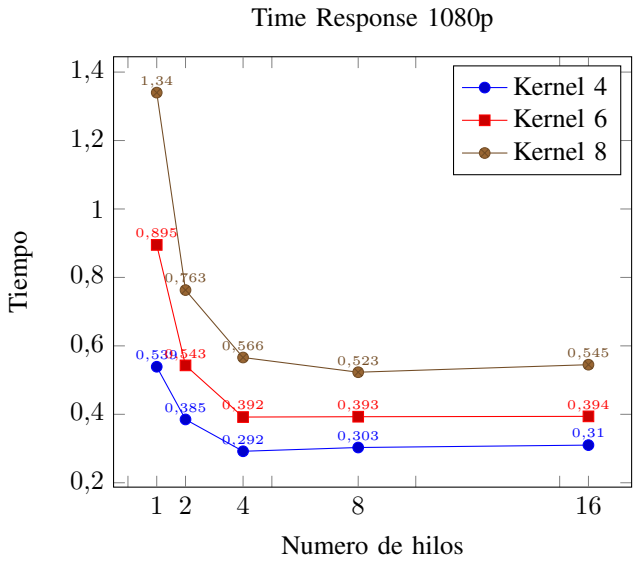
Como primera medida se utilizo el time response o tiempo de ejecución de un programa, donde para cada imagen se crearon sus corresponientes gráficas donde en eje horizontal se colocaron el numero de hilos que ejecutaba el programa y en el eje vertical el tiempo que tardo. Para no sobrecargar la gráfica se decidio incluir solo tres nucleos por gráfico, eso hace que por resolucion de imagen se crearan 2 graficos.
A continuacion se presentan las gráficas correspondientes:

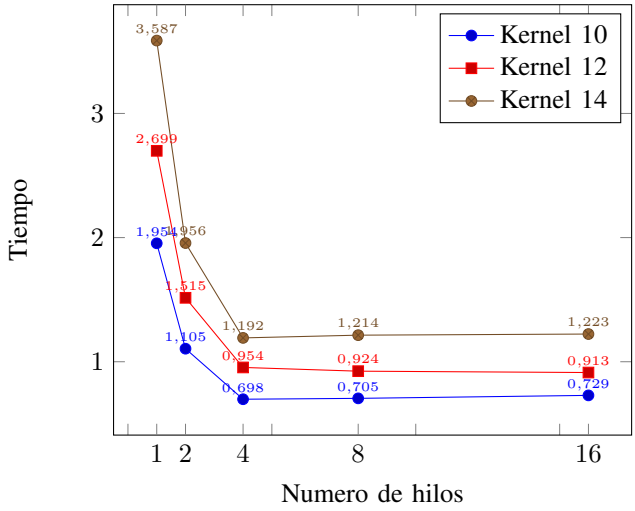
- **Imagen de 720p**



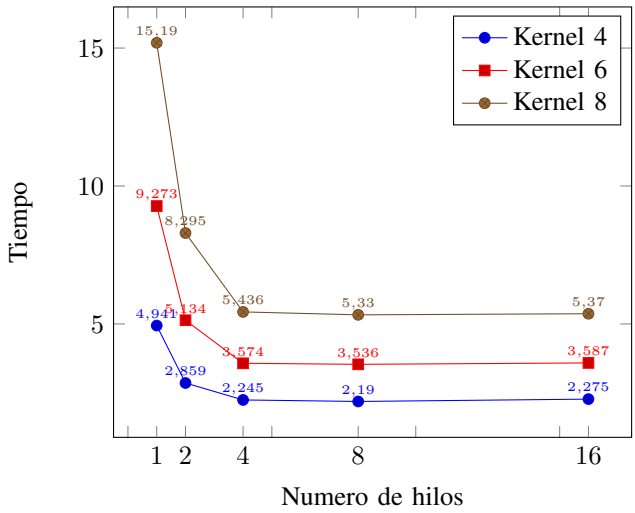


- **Imagen de 1080p**

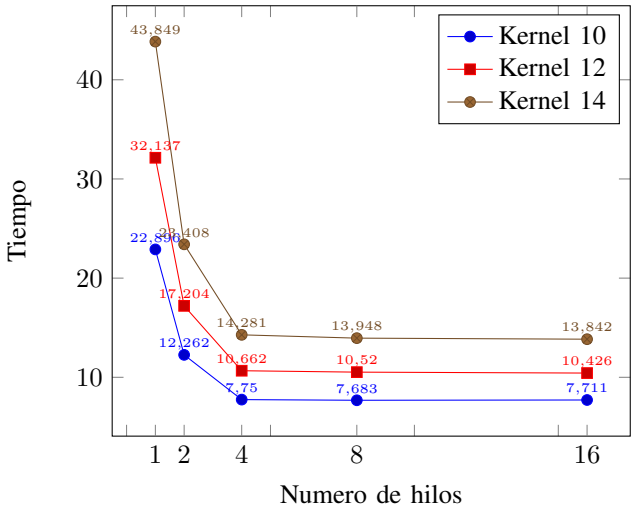




- **Imagen de 4k**



Time Response 4k

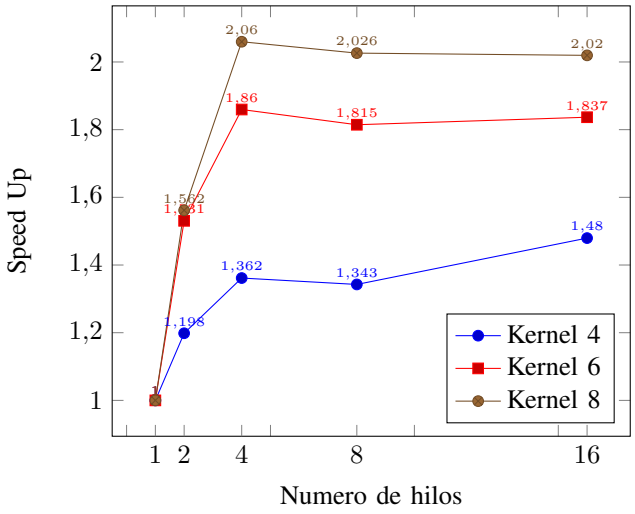


Como se puede observar en las gráficas anteriores el time response del código tiene cierta tendencia de mejorar conforme se aumenta la cantidad de hilos que son lanzados hasta los 4 hilos, de ahi en adelante la mejora es insignificante y en algunos casos como en la imagen de 720p se ve cierta desmejora.

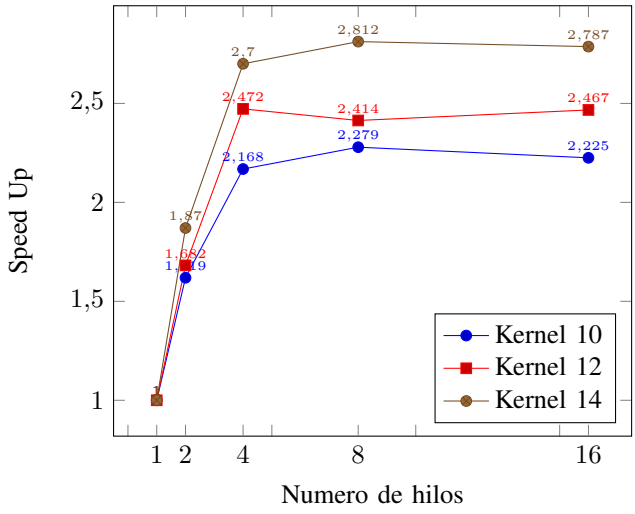
Otra medida que se considero determinar para medir el rendimiento de los hilos en el programa es el speed up es decir el tiempo de ejecución secuencial dividido entre el tiempo de ejecución por cada una de las cantidad de hilos(1,2,4,8,16). A continuación se presentan las gráficas correspondientes:

- **Imagen de 720p**

Speed Up 720p

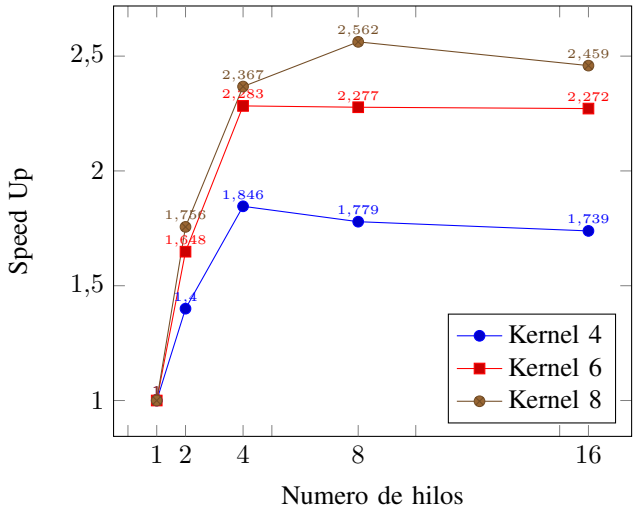


Speed Up 720p



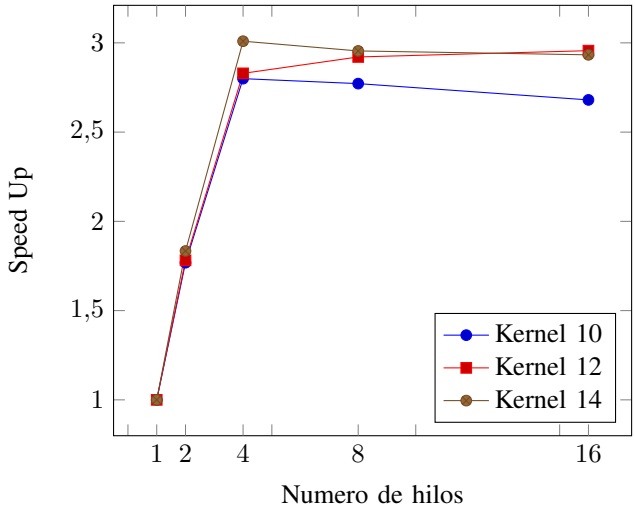
- **Imagen de 1080p**

Speed Up 1080p

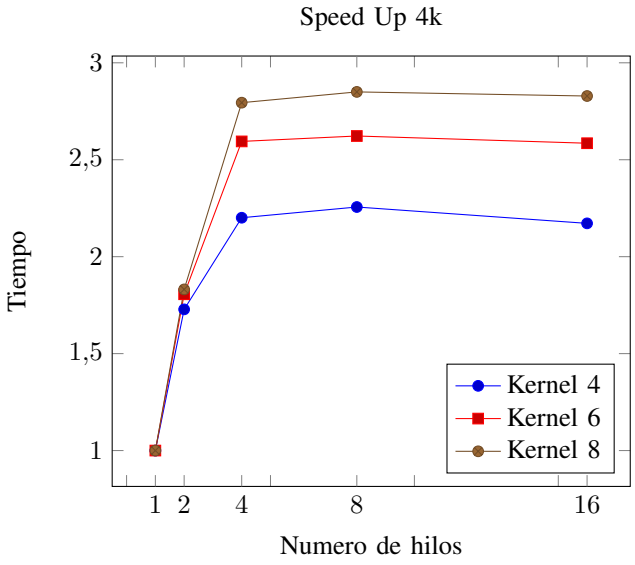


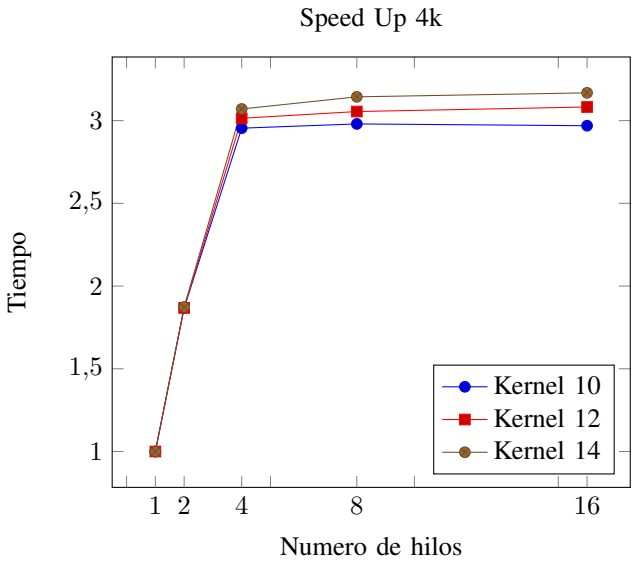
Para estas últimas tre gráficas se decidio omitir a proposito la leyenda del tiempo en cada uno de los nodos, debido a la superposicion que se genera.

Speed Up 1080p



- **Imagen de 4k**



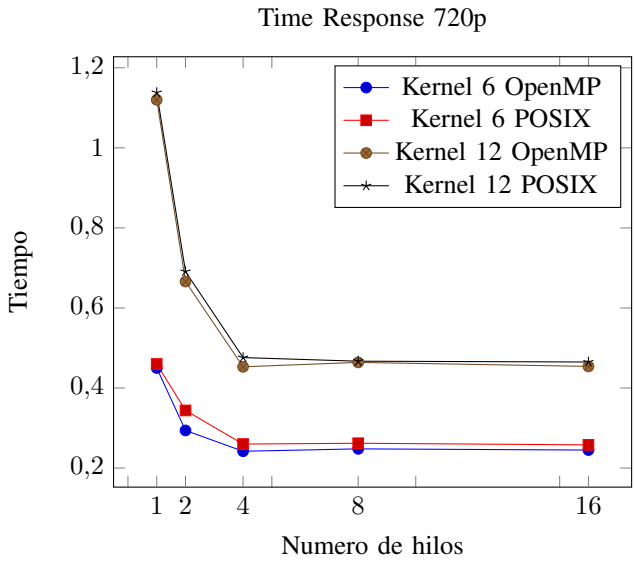


Como podemos observar en las gráficas anteriores el speed up es el esperado ya que el tiempo tiene una tendencia a aumentar conforme el programa se ejecuta con más hilos.

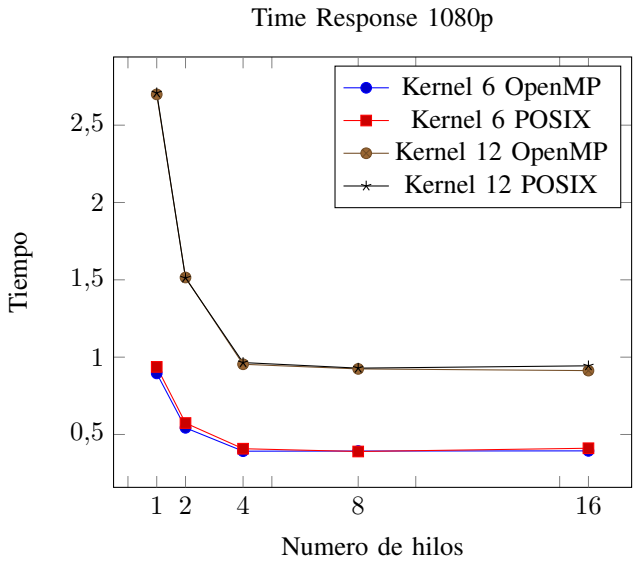
Por último se consideró importante realizar una comparación de time response entre el código de difuminado usando POSIX (descrito en la primera practica) y el actual que es usando la libreria OpenMP. Para simplificar un poco el informe se considero solamente comparar con los kernel 6 y 12 en cada una de las resoluciones de imagen. A continuación se presentan las gráficas correspondientes:

Para estas ultimas gráficas tambien se decidio omitir la leyenda del tiempo en cada uno de los nodos, debido a la superposición que se genera.

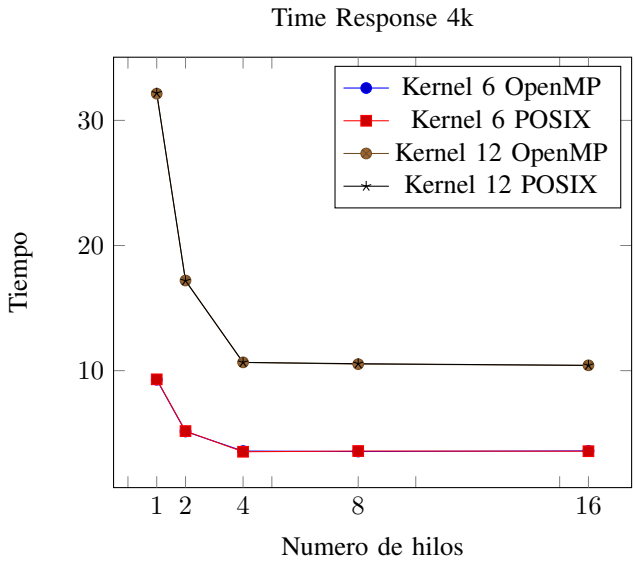
- **Imagen de 720p**



- **Imagen de 1080p**



- **Imagen de 4k**



Como se puede observar en las gráficas existe cierta mejora usando el equivalente de POSIX en OpenMp, aunque esta

mejora es más evidente en resolución de imagenes pequeñas como la de 720p.
La siguientes tablas ilustran lo anteriormente mencionado:

Diferencias entre POSIX Y OpenMP		
Hilos	**Kernel 6**	**Kernel 12**
1 Hilo	0,010	0,018
2 Hilos	0,050	0,025
4 Hilos	0,018	0,023
8 Hilos	0,014	0,003
16 Hilos	0,013	0,011
Promedio de diferencias	0,021	0,016

Cuadro II: Imagen de 720p

Diferencias entre POSIX Y OpenMP		
Hilos	**Kernel 6**	**Kernel 12**
1 Hilo	0,042	0,01
2 Hilos	0,031	-0,003
4 Hilos	0,016	0,011
8 Hilos	-0,003	0,005
16 Hilos	0,017	0,031
Promedio de diferencias	0,0206	0,0108

Cuadro III: Imagen de 1080p

Diferencias entre POSIX Y OpenMP		
Hilos	**Kernel 6**	**Kernel 12**
1 Hilo	0,042	0,01
2 Hilos	0,035	-0,024
4 Hilos	-0,055	0,011
8 Hilos	0,040	0,036
16 Hilos	-0,026	-0,008
Promedio de diferencias	0,0054	0,0082

Cuadro IV: Imagen de 4k

Con base en los datos tabulados se calculó la diferencia promedio entre la ejecución de POSIX y OpenMP en la paralelización del efecto difuminado. Para los kernel de 6 y 12 en el caso de la imagen de 720p es de 0,0185 segundos, en el caso de la imagen de 1080p es de 0,0157 segundos y en el caso de la imagen de 4k es de 0,0068 segundos. Por último cabe mencionar que la diferencia promedio global es de 0,01366666667 segundos.

CONCLUSIONES.

- Cuanto mas grande es el kernel con el que se trabaja se tarda un tiempo superior en realizar el efecto difuminado debido a que realiza mas operaciones al momento de calcular el promedio del kernel, haciendo que el costo computacional sea superior.
- Como se observa en los resultados presentados en las tablas anteriores la mejora solo se aprecia que es significativa hasta los 4 hilos y considero que se debe a que el procesador donde se corrio el código solo posee 4 núcleos
- Aunque la diferencia entre el uso de POSIX y OpenMP podria parecer muy pequeña en realidad al momento de hacer un gran conjunto de operaciónes que requieran de un cómputo mas prolongado los tiempos tenderán a escalar.
- Cabe mencionar que la implementación de OpenMP es muy sencilla incluso cuando el programa es secuencial, lo que va a permitir migrar a una computacion paralela rápidamente cuando existe alta densidad de código.

REFERENCIAS

[1] Pedraza, C. (2019). Diapositivas de clase.
[2] David, K. and Wen-Mei, H. (2010). Programming massively parallel processos. 2nd ed. Morgan Kaufmann.
[3] Thomas, R. and Gudula, R. (2010). Parallel Programming. 1st ed. Springer.
[4] En.wikipedia.org. (2019). $Boxblur. Availableat : https : //en.wikipedia.org/wiki/Box_blur.$