Prácticas: Difuminado de imagen

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

ÍNDICE

***Resumen*—El presente informe muestra como se llevo a cabo la paralelización mediante el uso de distintas plataformas de programación como lo son POSIX, openMP y CUDA del efecto difuminado.Para un correcto analisis de los resultados se tuvo en cuenta como metricas tanto el Speed Up como de Time Response para los distintas resoluciones de imagenes (720p, 1080p y 4k)), de kernels (2,3,4,5, 6,7, 8,9, 10,11 ,12 13 y 14) y de hilos (1,2,3,4,5,6,7,8,9 y 10)**

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 de 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

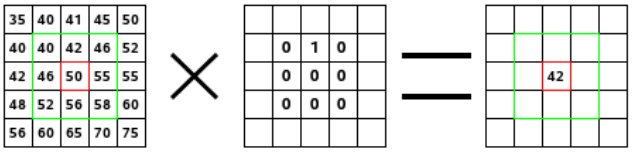


Figura 1: Convolucion matricial

I-A. Creación del efecto difuminado

Mas especificamente se utilizo 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 promedio que hay en los promedios de la matriz 5x5 cuyo centro es el pixel en cuestion.

II. Algoritmo de paralización.

La paralelizacion se realizo mediante asignación tipo block-wise es decir que se tomo la imagen y se dividio a lo alto en filas y cada una de estas se asigno a un hilo. Es decir que un caso hipotetico donde se quiera aplicar el efecto difuminado a una imagen con un alto de 368 pixeles,se lanzan 4 hilos donde el hilo 1 se le asignaba la fila 1(del pixel 0 al 92), al 2 la fila 2(del pixel 92 al 184), al 3 la columna 3(del pixel 184 al 276) y al 4 la columna 4(del pixel 276 al 368) . Para mayor clarificación detras de la idea que se uso para la paralización, la siguiente imagen:

Figura 2: Algoritmo de paralelización

El ejemplo anterior muestra el caso en el cual se lanza un solo bloque con 4 hilos. Lo mismo se realiza en las otras tecnicas de paralelización y de computacion distribuida sino en vez de hilos se usan bloques, procesos,nodos y/o una mezcla

III. Experimentos y resultados.

III-A. POSIX y OpenMP

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

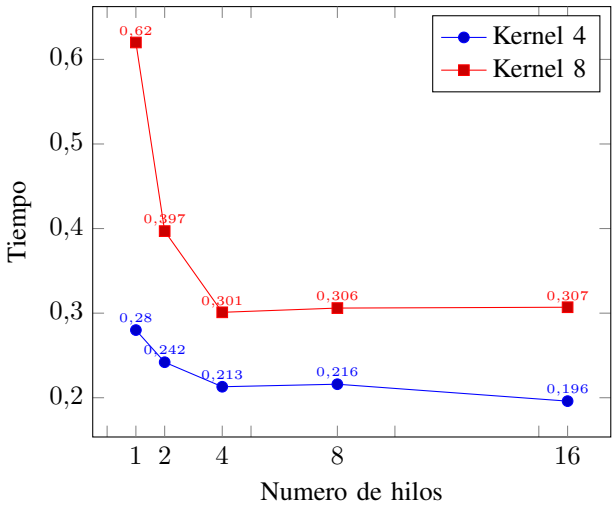
Cuadro I: Especificaciones del ordenador

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

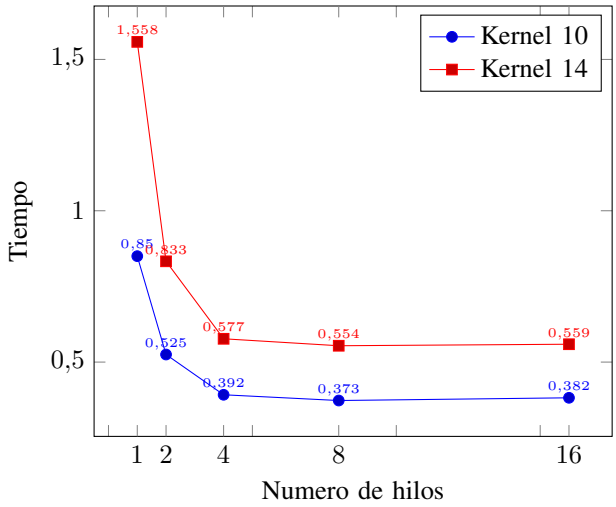
III-A1. Resultados: Como primera medida se utilizo el time response o tiempo de ejecución de un programa, donde para cada imagen se crearon sus corresponientes gráficas donde en eje horizontal se colocaron el numero de hilos que ejecutaba el programa y en el eje vertical el tiempo que tardo. Para no sobrecargar la gráfica se decidio incluir solo tres nucleos por grafico, eso hace que por resolucion de imagen se crearan 2 graficos.
A continuacion se presentan las gráficas correspondientes:

- **Imagen de 720p**

Time Response 720p

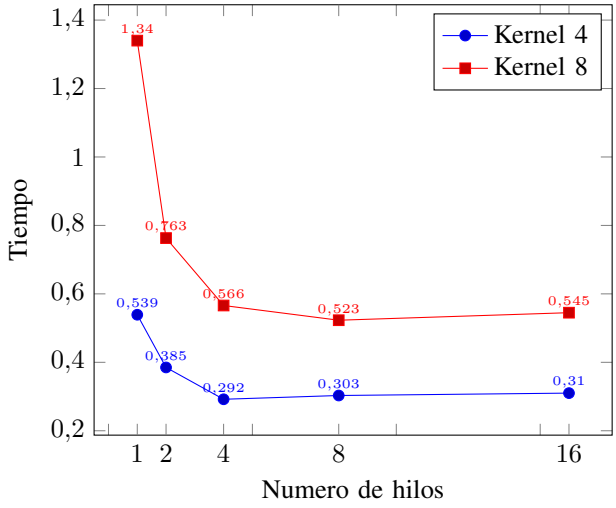


Time Response 720p

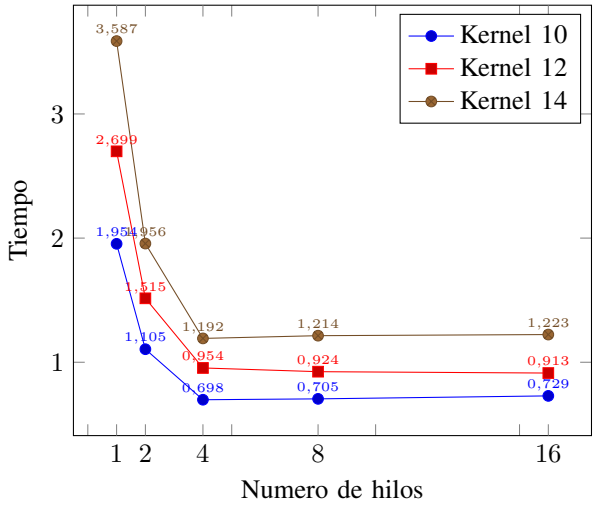


- **Imagen de 1080p**

Time Response 1080p

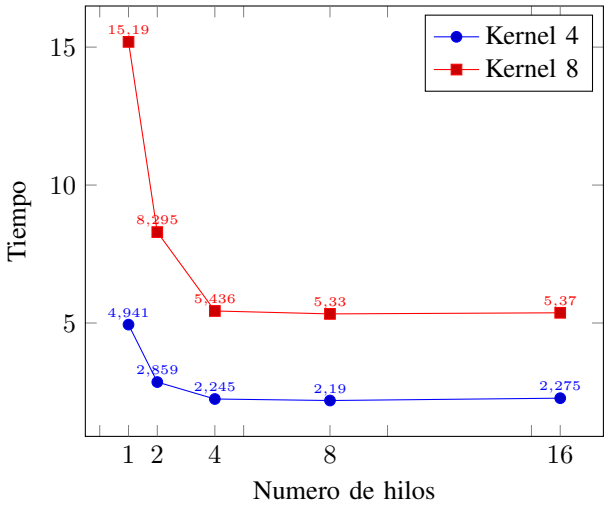


Time Response 1080p

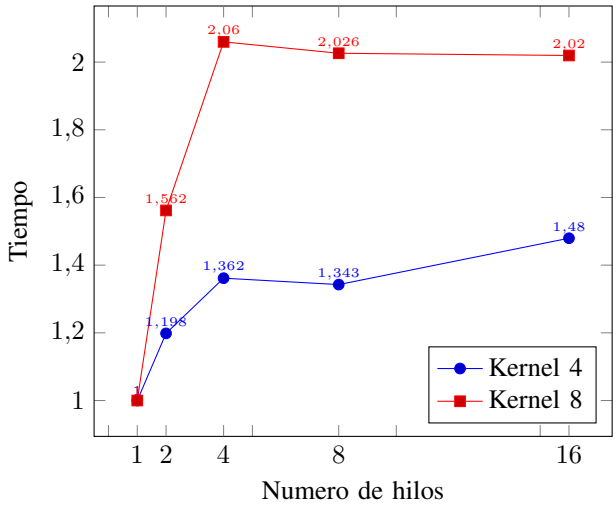


- **Imagen de 4k**

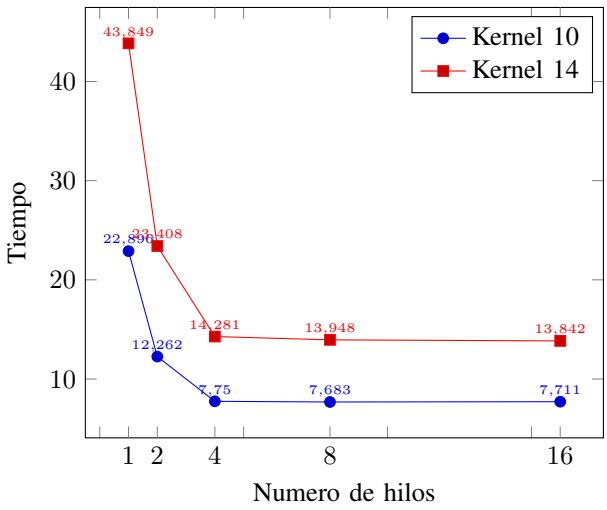
Time Response 4k



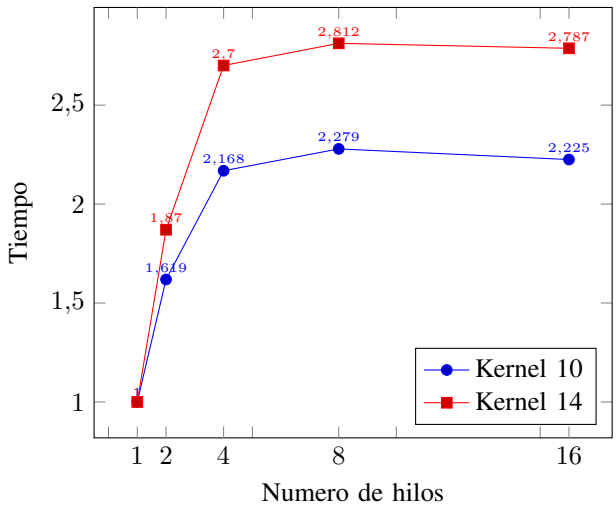
Speed Up 720p



Time Response 4k



Speed Up 720p

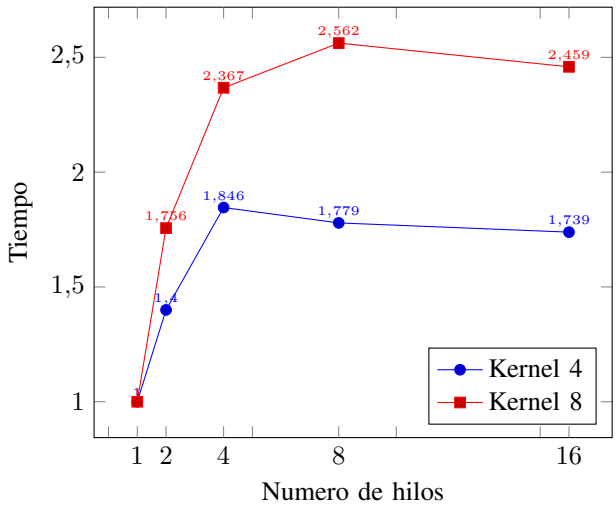


Como se puede observar en las gráficas anteriores el time response del código tiene cierta tendencia de mejorar conforme se aumenta la cantidad de hilos que son lanzados hasta los 4 hilos, de ahi en adelante la mejora es insignificante y en algunos casos como en la imagen de 720p se ve cierta desmejora.

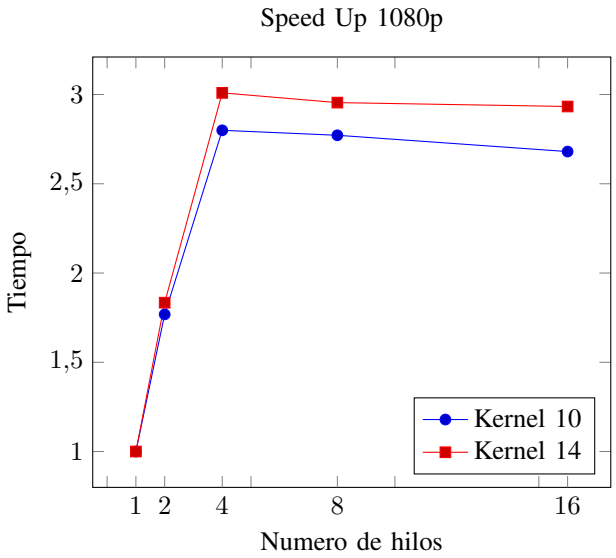
Otra medida que se considero determinar para medir el rendimiento de los hilos en el programa es el speed up es decir el tiempo de ejecución secuencial dividido entre el tiempo de ejecución por cada una de las cantidad de hilos(1,2,4,8,16). A continuación se presentan las gráficas correspondientes:

- **Imagen de 720p**
- **Imagen de 1080p**

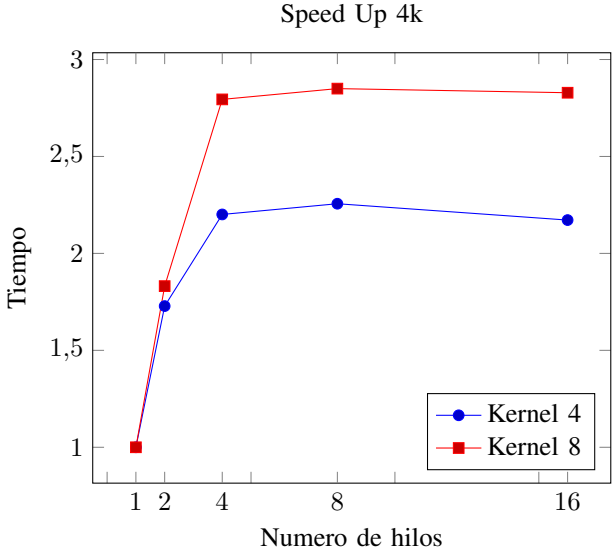
Speed Up 1080p

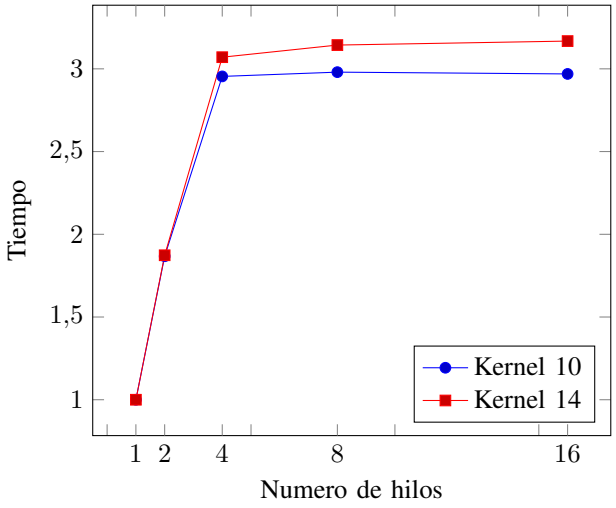


Para estas últimas tre gráficas se decidio omitir a proposito la leyenda del tiempo en cada uno de los nodos, debido a la superposicion que se genera.



- **Imagen de 4k**



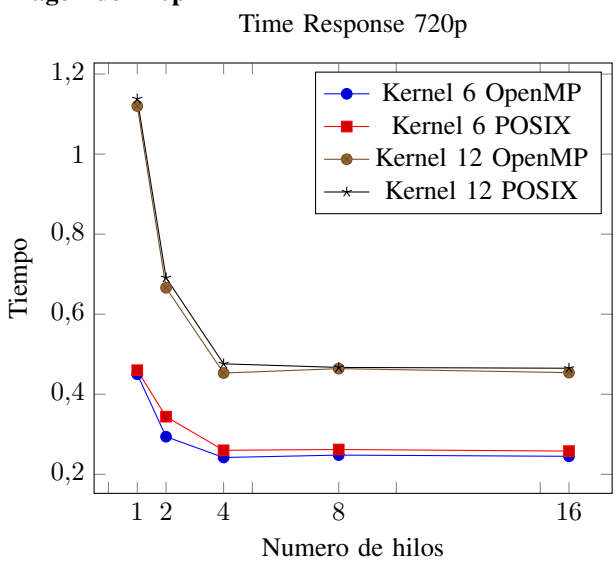


Como podemos observar en las gráficas anteriores el speed up es el esperado ya que el tiempo tiene una tendencia a aumentar conforme el programa se ejecuta con más hilos.

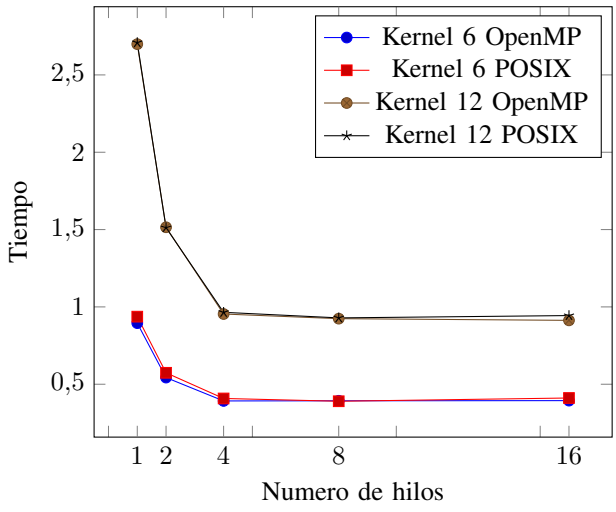
Por último se considero importante realizar una comparación de time response entre el código de difuminado usando POSIX que fue el de la primera practica y el actual que es usando la librerio OpenMP. Para simplificar un poco el informe se considero solamente comparar con los kernel 6 y 12 en cada una 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**

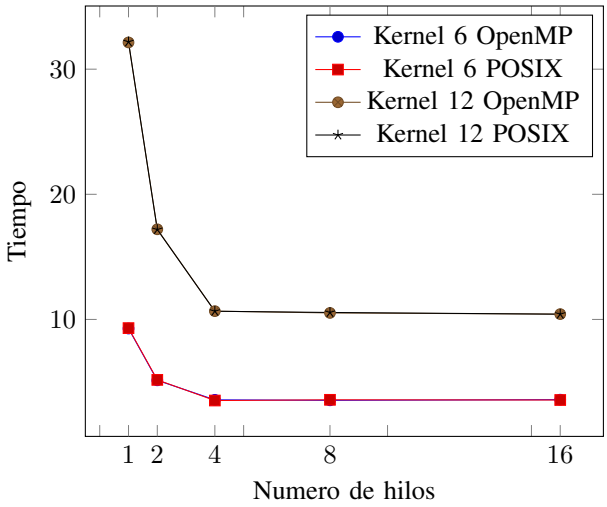


- **Imagen de 1080p**



- **Imagen de 4k**

Time Response 4k



Como podemos 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

En base a las anteriores tablas se calcula 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

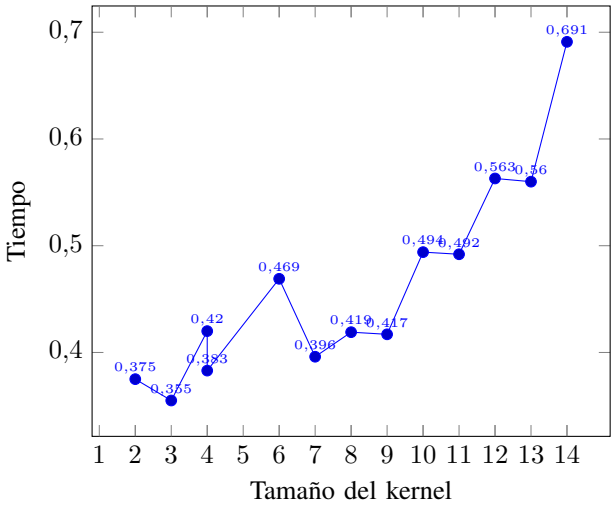
III-B. CUDA

Antes de mostrar los resultados obtenidos y compararlos con los que se obtuvieron al paralelizar el efecto difuso que se obtuvo con CUDA en comparación con los de CPU(POSIX Y OpenMP), es necesario decir que se corrio en el entorno de Google Colab.

III-B1. 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 tamaño del kernel que ejecutaba el programa y en el eje vertical el tiempo que tardo.
A continuacion se presentan las gráficas correspondientes:

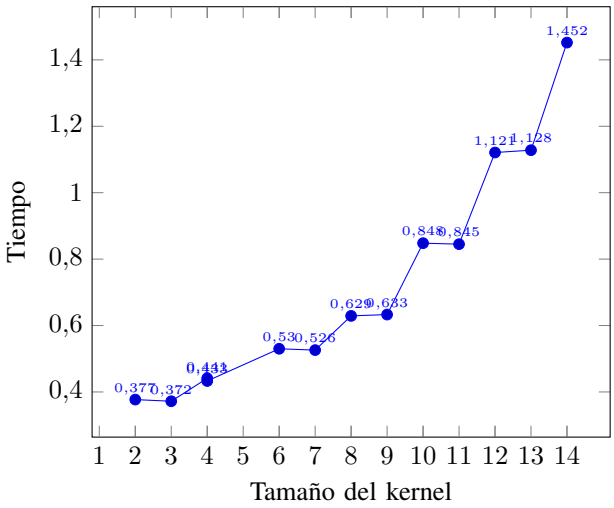
- **Imagen de 720p**

Time Response 720p

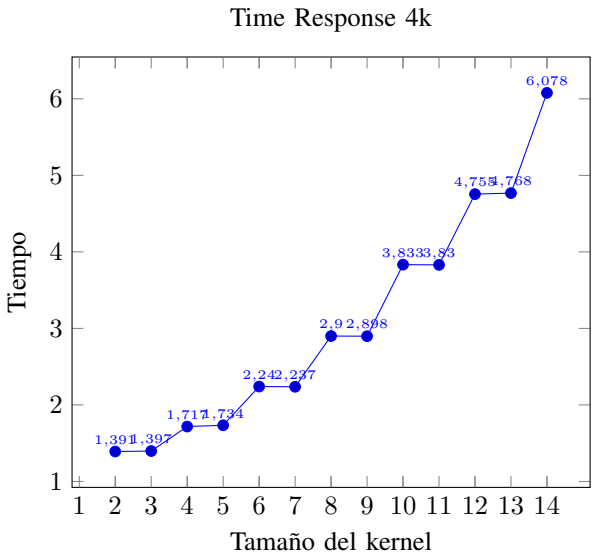


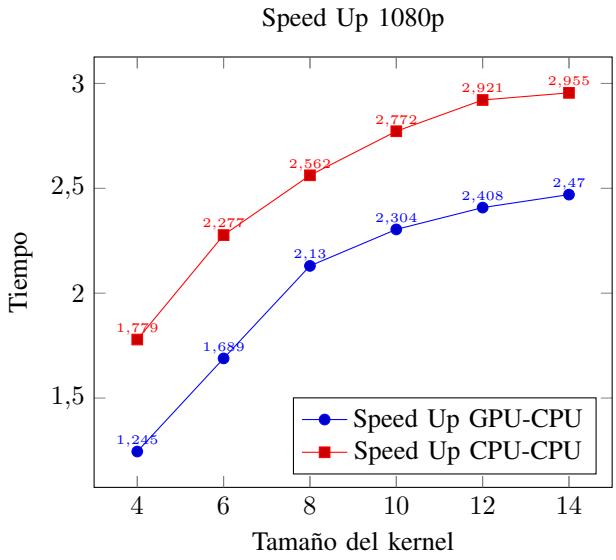
- **Imagen de 1080p**

Time Response 1080p

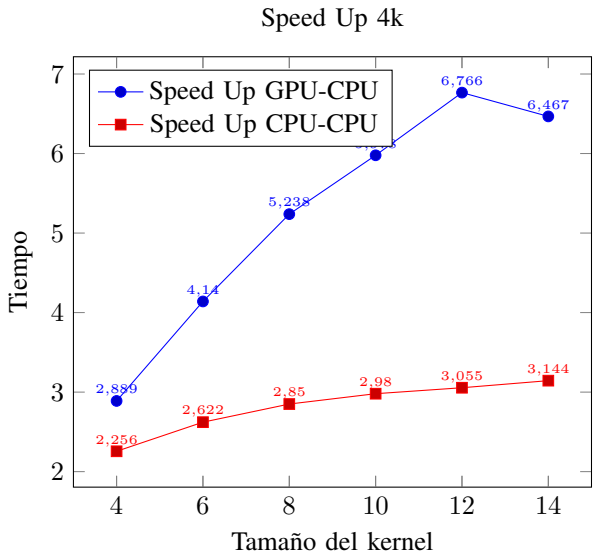


- **Imagen de 4k**



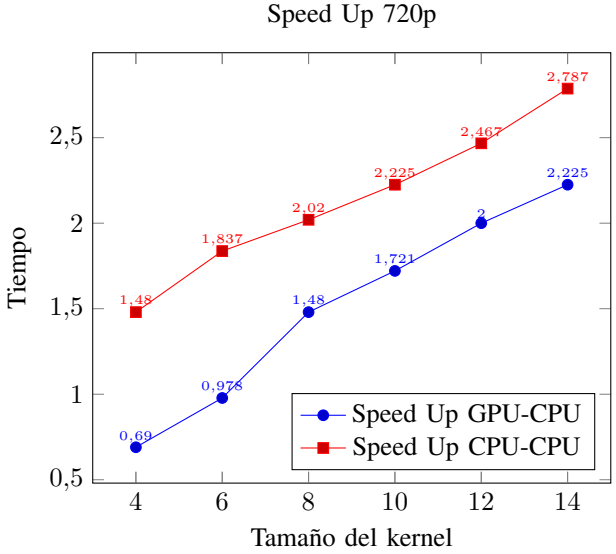


- **Imagen de 4k**



Como se puede observar en las gráficas anteriores el time response del código tiene cierta tendencia de empeorar conforme se aumenta el tamaño del kernel, como es el comportamiento esperado.
Otra medida que se considero determinar para medir el rendimiento de los hilos en el programa es el speed up que para este caso se calculo como la división entre el mejor tiempo en CPU contra con los mejores tiempos en GPU
A continuación se presentan las gráficas correspondientes:

- **Imagen de 720p**

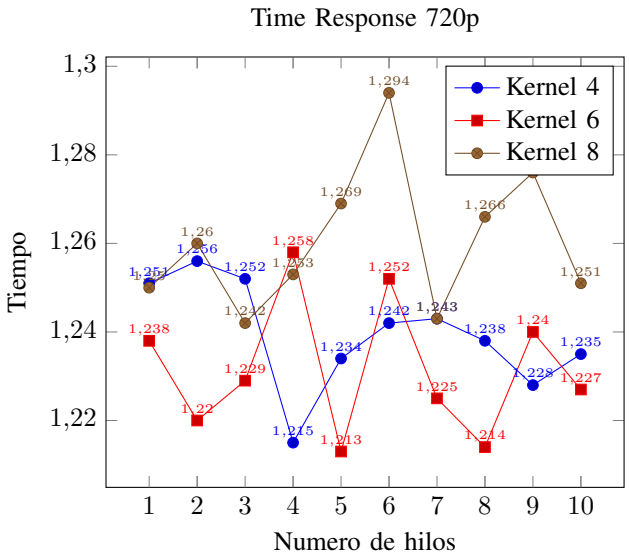


- **Imagen de 1080p**

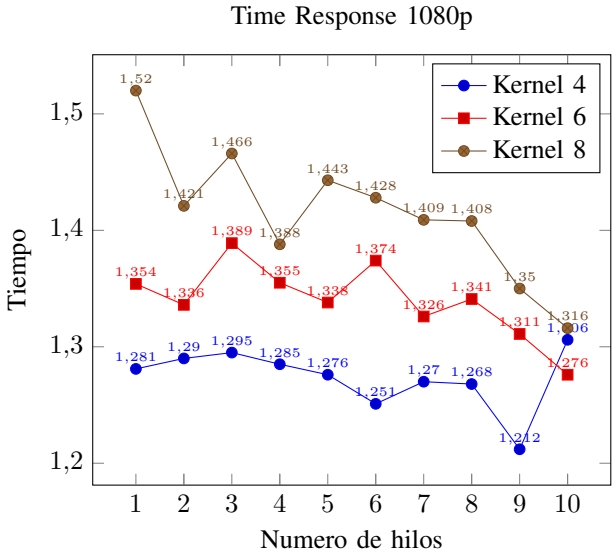
III-C. OpenMPI

Para esta practica se utilizo 2 clausters, uno que fue creado en google cloud el cual tiene 5 maquinas con las mismas configuraciones es decir con 7.5 gb de ram y 2 vcpu y otro en microsoft azure con 3.5 gb de ram y 1 solo vcpu. El algoritmo para distribuir entre maquinas utiliza MPI indicando un proceso por cada máquina del cluster(5 máquinas) y variando el numero de hilos por máquina(1 - 10) . Se corrió el algoritmo y se sacaron las siguientes gráficas en el caso del time response:

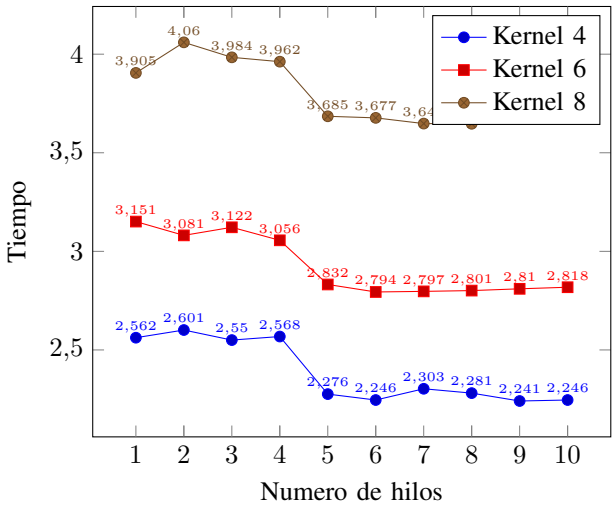
- **Imagen de 720p**



- **Imagen de 1080p**

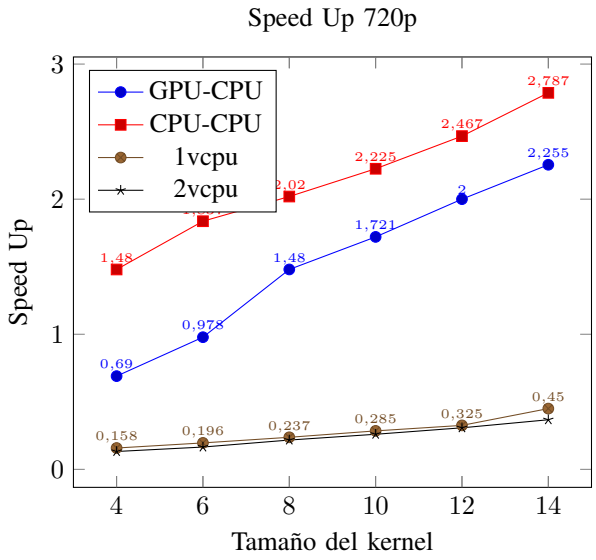


- **Imagen de 4k**

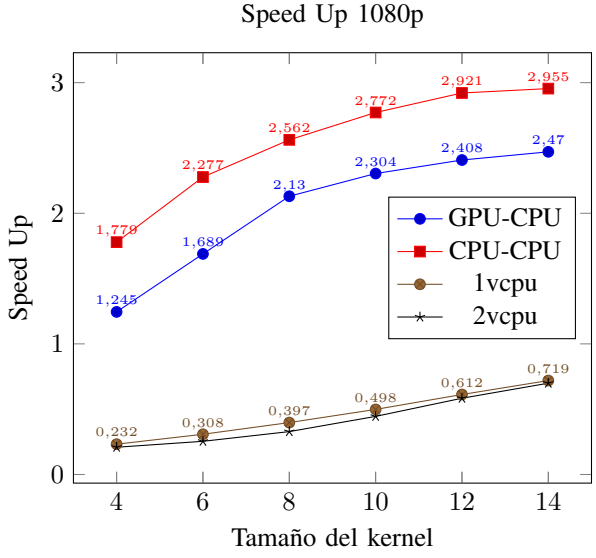


Al momento de graficar el speed up se comparo el de la anterior practica que fue la de CUDA con el speed up de la practica actual. Es decir se hizo la comparación entre GPU-CPU, CPU-CPU, 1 vcpu y 2 vcpu. Las gráficas obtenidas para los diferentes tamaños de imagenes se muestran a continuación:

- **Imagen de 720p**

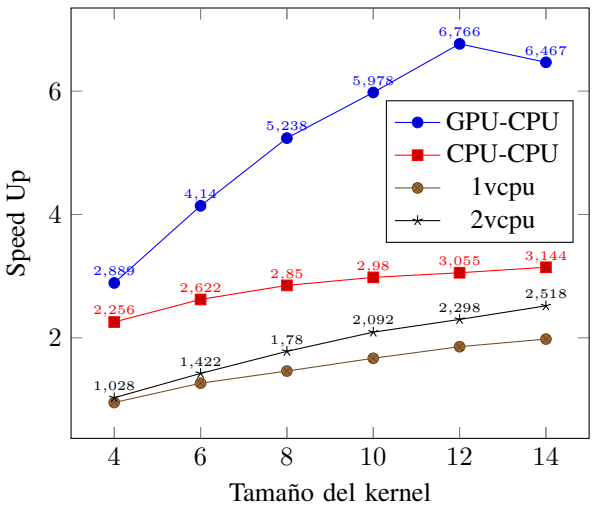


- **Imagen de 1080p**



- **Imagen de 4k**

Speed Up 4k



IV. Conclusiones

1. POSIX y OpemMP
   - Entre mas grande es el kernel con el que se trabaja se demora 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 nucleos
   - 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 tipo difuminado esta puede ser considerable. Tambien hay que considerar que el algoritmo a nuestra consideración esta bastante optimizado como para que haya una diferencia mayor entre el uso de POSIX y OpenMP
2. CUDA
   - Entre mas grande es el kernel con el que se trabaja se demora 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
   - Se evidencia que el speedup de GPU es menor cuando el computo a realizar es considerablemente menor, ya que es necesario tener en cuenta los tiempos de copia de los datos a calcular en el device y luego la recepcion en host, además que para la implementacion en cuestión es necesario realizar una transformacion de matrizde OpenCV a arreglo de enteros.
   - Es necesario realizar una comparacion de las implementaciones de CPU y GPU para reconocer cual tiene mejor rendimiento, para nuestra práctica solo vale la pena usar GPU en el computo de imagenes 4K.
3. OpenMPI
   - Los equipos que computan el problema son maquinas virtuales que están asignadas por google cloud, esto no necesariamente implica que las maquinas son contiguas en una LAN, lo que se asegura es que hacen parte de una misma VLAN lo cual impacta en el desempeño.
   - Asegurar que todas las maquinas en el cluster tienen las mismas librerias instaladas no siempre puede ser trivial, lo cual dificulta la implementación.
   - al igual que en GPU la aceleracion aumenta en la medida que el tiempo de cómputo es cosiderablemente mayor al tiempo de transporte de la información.
   - La grafica de speedup sufre un cambio similar al de la implementación en GPU, pero ya que la aceleracion es pequeña en comparación no es tan notoria la pendiente positiva.
   - Es evidente que existe una aceleración de los tiempos de respuesta con respecto a la mejor implementación paralela en una sola máquina, sin embargo aun no se iguala la velocidad que alcanzó la implementación en una GPU.

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). $Boxblur. Availableat : https : //en.wikipedia.org/wiki/Box_blur$.