

Práctica: Multiplicación de matrices

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

ÍNDICE

I. Introducción.	1
I-A. Multiplicación matricial	1
II. Algoritmo del paralelización.	1
III. Experimentos y resultados.	1
III-A. POSIX y OpenMP	1
III-A1. Resultados	2
III-B. CUDA	4
III-B1. Resultados	4
III-C. OpenMPI	6
IV. Conclusiones	7
Referencias	8

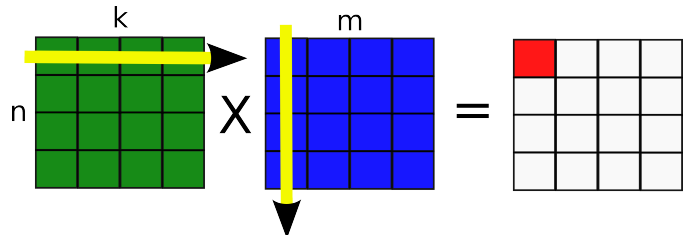


Figura 1: Multiplicación matricial

I-A. Multiplicación matricial

Para esta práctica se paraleliza el algoritmo naive de multiplicación de matrices. Donde se tomaron las matrices de archivos csv donde se prepararon matrices aleatorias de diferente tamaño. Cabe aclarar que el nombre indica el tamaño de la matriz. Es decir si el archivo se llama 16A.csv eso quiere decir que es la primera matriz y tiene un tamaño de 16x16.

II. ALGORITMO DEL PARALELIZACIÓN.

La paralelización se realizó mediante asignación tipo blockwise es decir que se tomó la primera matriz y se dividió a lo alto en filas y a cada una de estas se asignó a un hilo o bloque. Es decir que un caso hipotético donde se quiera hacer la multiplicación a dos matrices de tamaño 16, y se lanzan 4 hilos entonces el hilo 1 se le asignaba la fila 1 a la 4, al 2 la fila 5 a la 8, al 3 la fila 9 a la 12 y al 4 de la fila 13 a la 16.

Cada hilo hace la multiplicación correspondiente generando en la matriz producto la fila que le corresponde

I. INTRODUCCIÓN.

La multiplicación de matrices es una de las operaciones más básicas que incluso es enseñada en la primaria. Pero a pesar de su simplicidad ha sido altamente estudiada en el campo de la computación lo cual ha hecho que se desarrollen algoritmos tanto secuenciales como paralelos para resolver el problema de una manera más óptima. Dentro de estos podemos mencionar el de Strassen, divide and conquer, Coppersmith-Winograd entre otros

Aun así paralelizar tiene su límite el cual 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.

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 corrió el algoritmo. Para esto se construyó la siguiente tabla resumiendo dicha información:

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

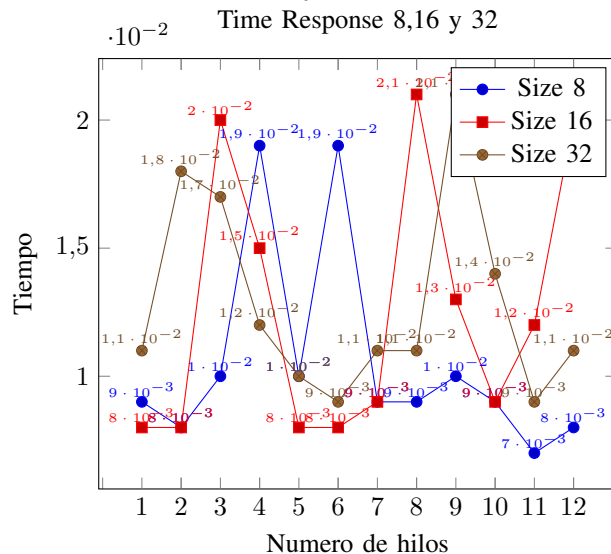
Cuadro I: Especificaciones del ordenador

Para evitar o disminuir posibles interferencias se corrió un script que automaticamente 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 multiplicación matricial se crearon sus correspondientes 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 tamaños de matriz por grafico.

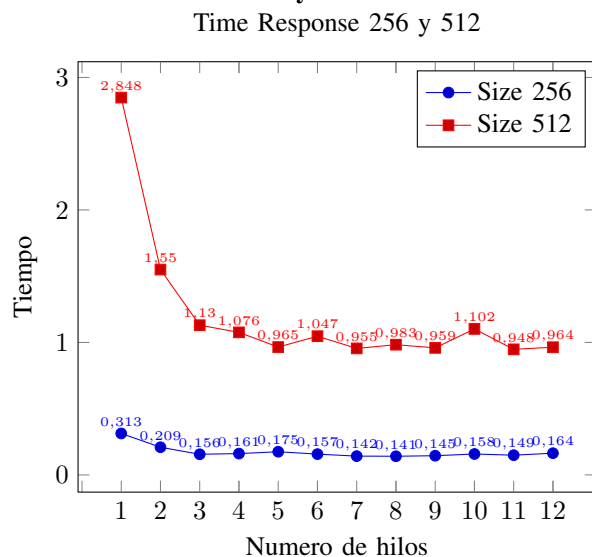
A continuacion se presentan las gráficas correspondientes:

■ Matrices de tamaño 8,16 y 32

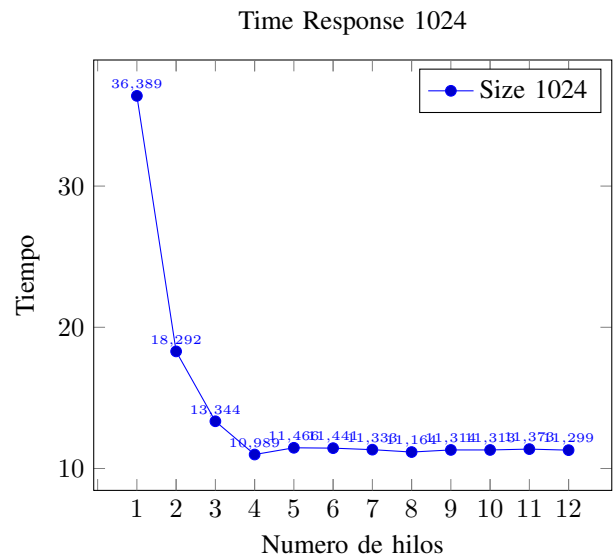


Como podemos observar de la anterior gráfica el beneficio de la paralelización en matrices pequeñas no es significativo. Esto se debe al tiempo de lanzado de hilos en estos casos aun es un tiempo significativo si lo comparamos con el tiempo total que demora el algoritmo.

■ Matrices de tamaño 256 y 512



■ Matriz de tamaño 1024



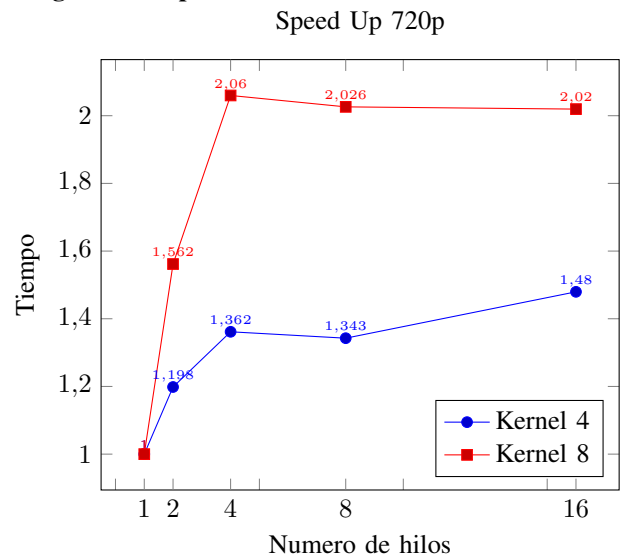
Como podemos ver de las anteriores 2 gráficas el comportamiento de la paralelización tiende a ser el esperado en matrices grandes donde el tiempo de lanzamiento de los hilos ya no es significativo.

Tambien se observa que el time response tiene cierta tendencia a mejorar conforme se aumenta la cantidad de hilos que son lanzados hasta los 4 hilos, de ahí en adelante la mejora es insignificante y en algunos casos tiende a desmejorar. Esto se debe a que el computador donde se corrio el código solo tiene 4 nucleos

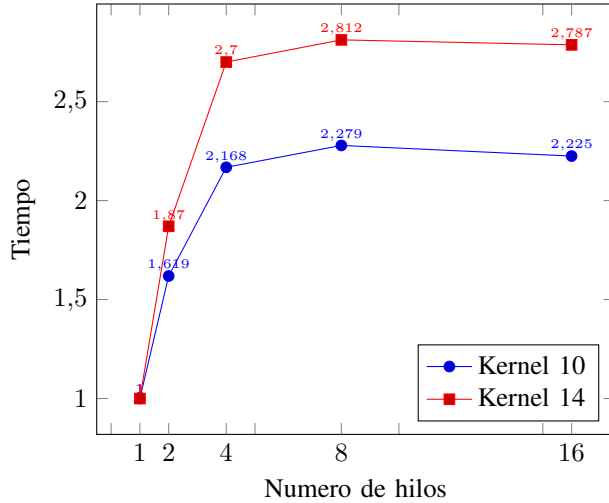
Otra medida que se considero determinar para medir el rendimiento del programa es el speed up es decir el tiempo de ejecución secuencial dividido entre el tiempo de ejecución paralelo.

A continuación se presentan las gráficas correspondientes:

■ Imagen de 720p

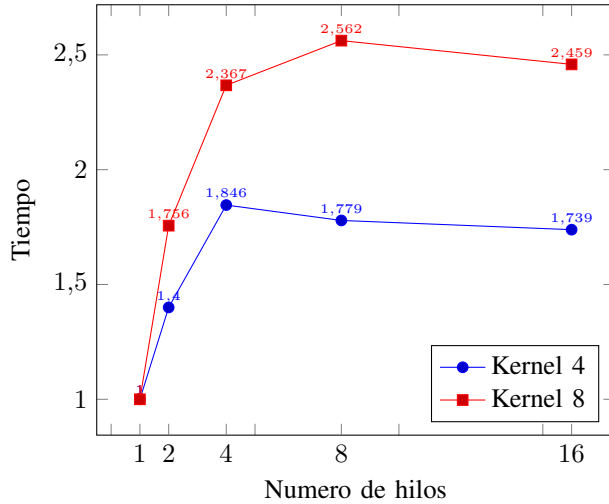


Speed Up 720p



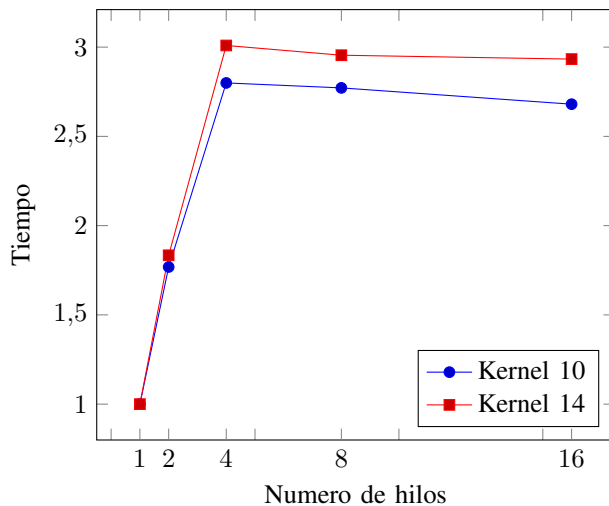
■ Imagen de 1080p

Speed Up 1080p



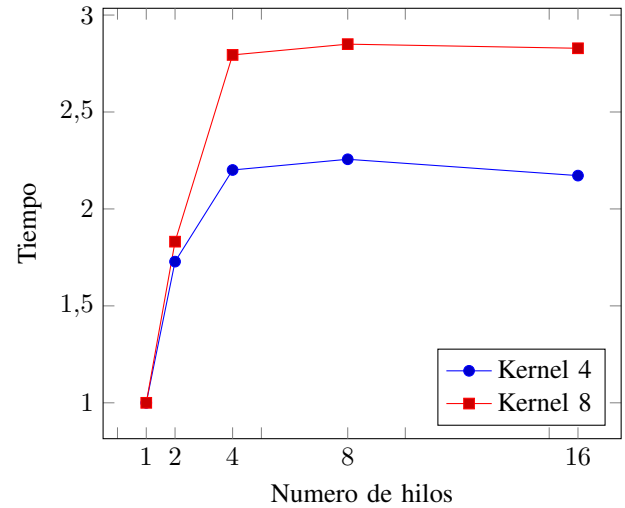
Para estas últimas tre gráficas se decidió omitir a proposito la leyenda del tiempo en cada uno de los nodos, debido a la superposicion que se genera.

Speed Up 1080p

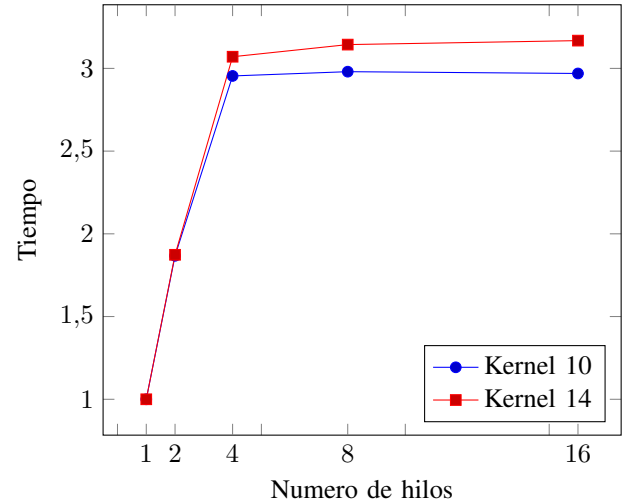


■ Imagen de 4k

Speed Up 4k



Speed Up 4k



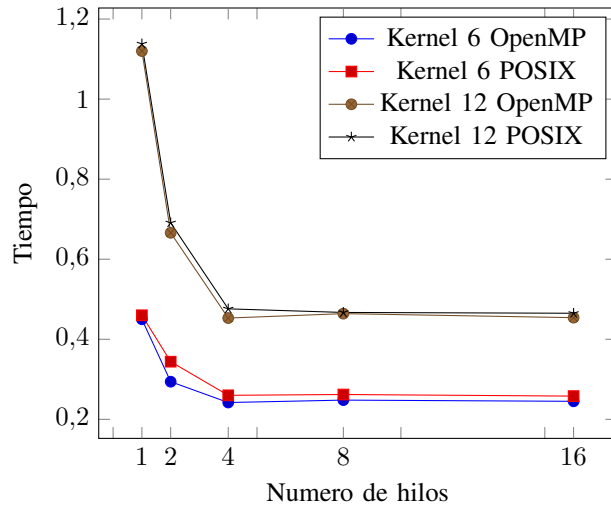
Como podemos observar en las gráficas anteriores el speed up es el esperado ya que el tiempo tiene una tendencia a aumentar conforme el programa se ejecuta con más hilos.

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 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 decidió omitir la leyenda del tiempo en cada uno de los nodos, debido a la superposición que se genera.

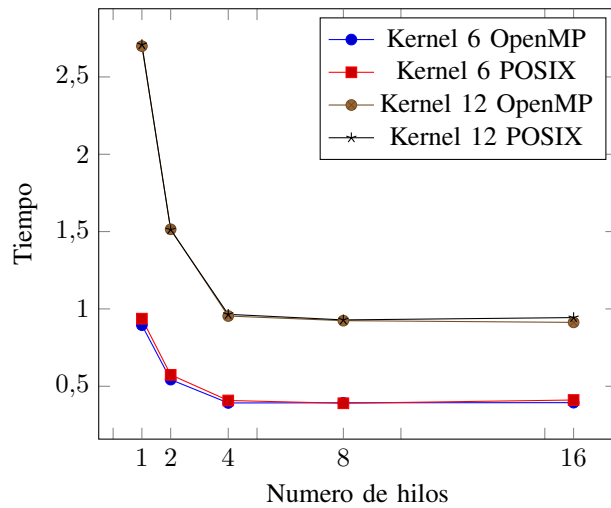
■ Imagen de 720p

Time Response 720p



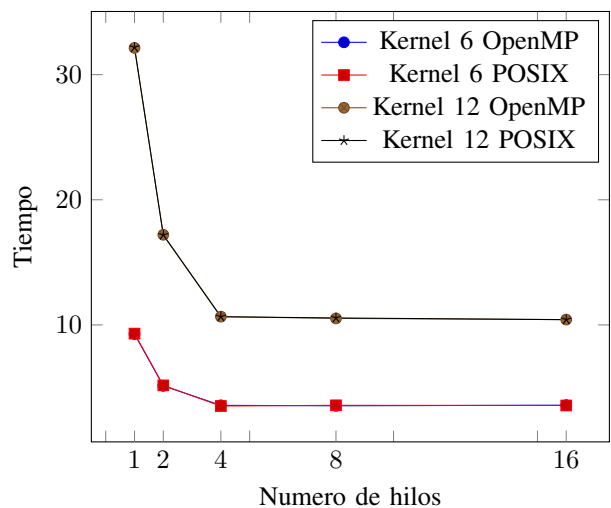
■ Imagen de 1080p

Time Response 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

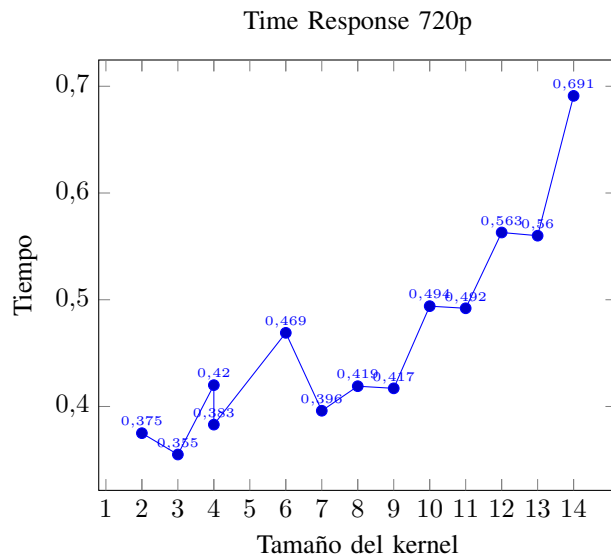
III-B. CUDA

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 corrió en el entorno de Google Colab.

III-B1. Resultados: Como primera medida se utilizó el time response o tiempo de ejecución de un programa, donde para cada imagen se crearon sus correspondientes gráficas donde en eje horizontal se colocaron el tamaño del kernel que ejecutaba el programa y en el eje vertical el tiempo que tardó.

A continuación se presentan las gráficas correspondientes:

■ Imagen de 720p

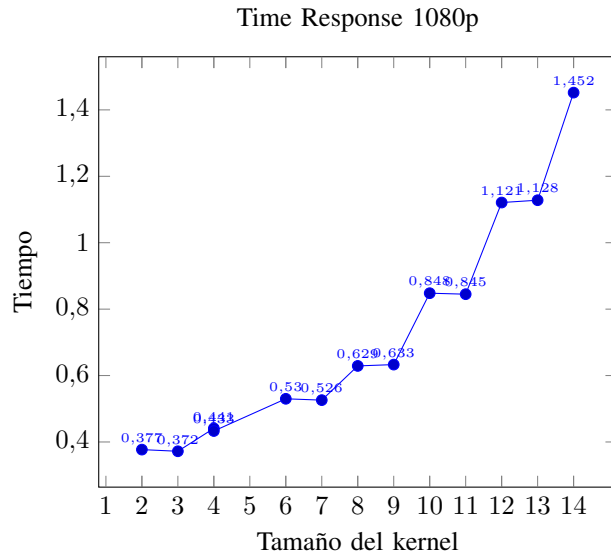


conforme se aumenta el tamaño del kernel, como es el comportamiento esperado.

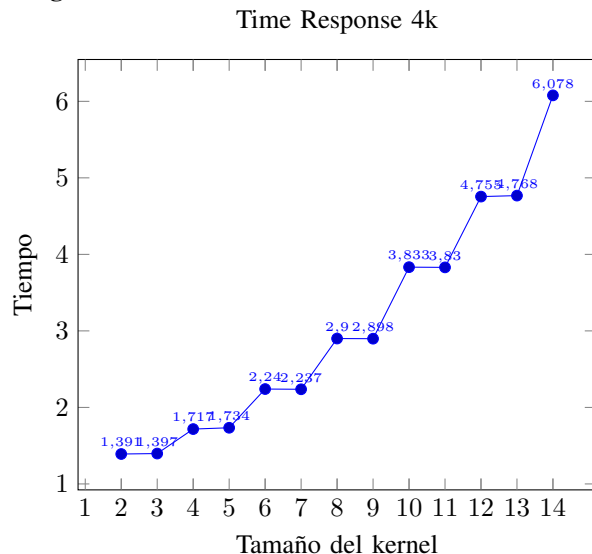
Otra medida que se considero determinar para medir el rendimiento de los hilos en el programa es el speed up que para este caso se calculo como la división entre el mejor tiempo en CPU contra con los mejores tiempos en GPU

A continuación se presentan las gráficas correspondientes:

■ Imagen de 1080p

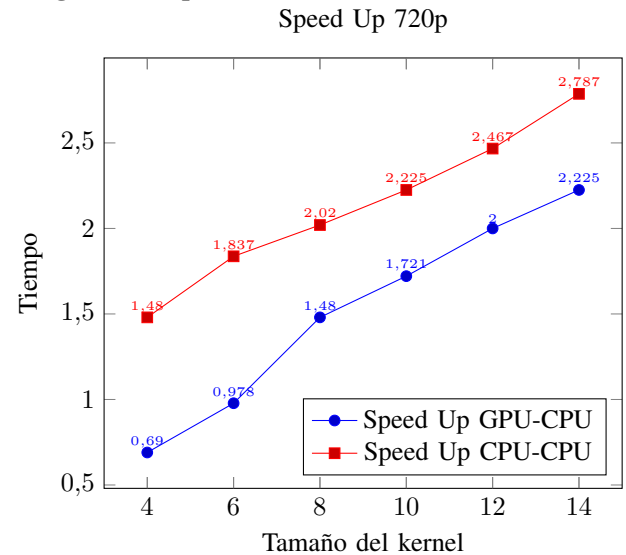


■ Imagen de 4k

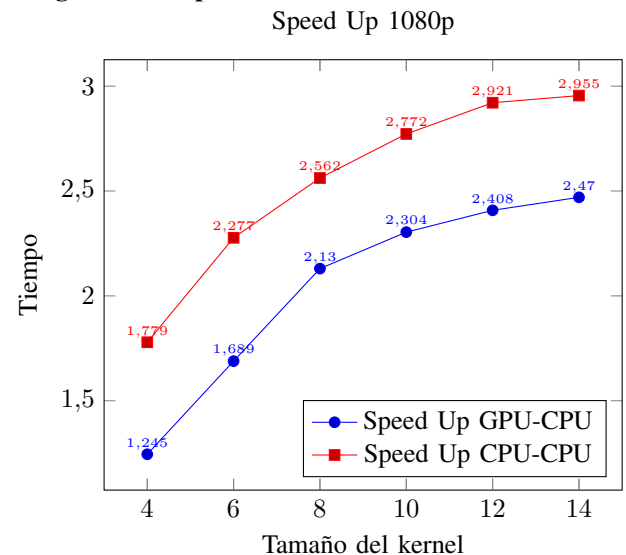


Como se puede observar en las gráficas anteriores el time response del código tiene cierta tendencia de empeorar

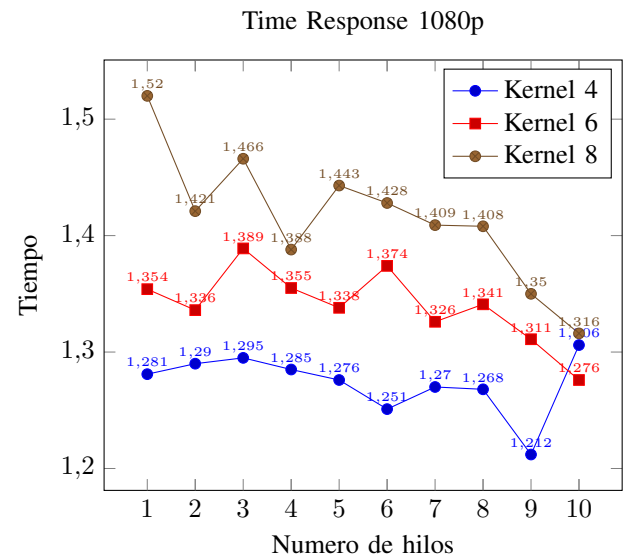
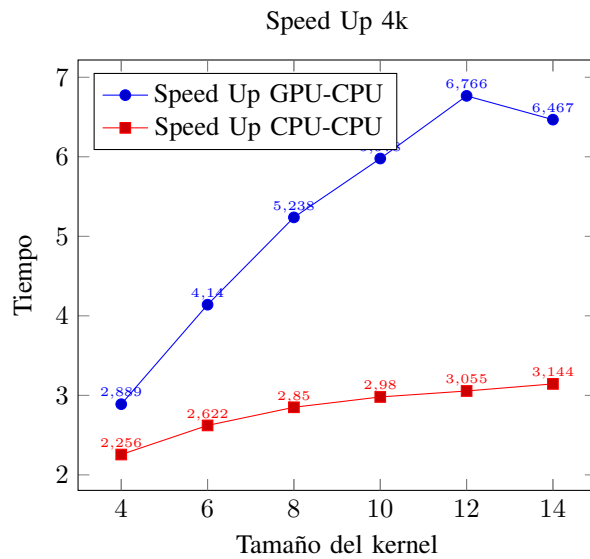
■ Imagen de 720p



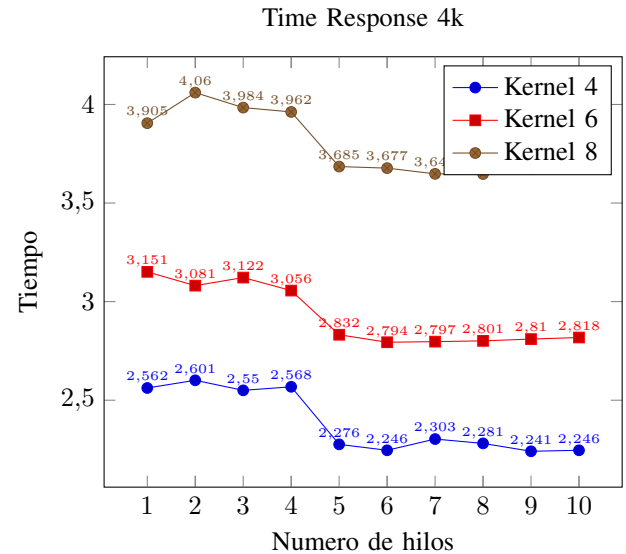
■ Imagen de 1080p



■ Imagen de 4k



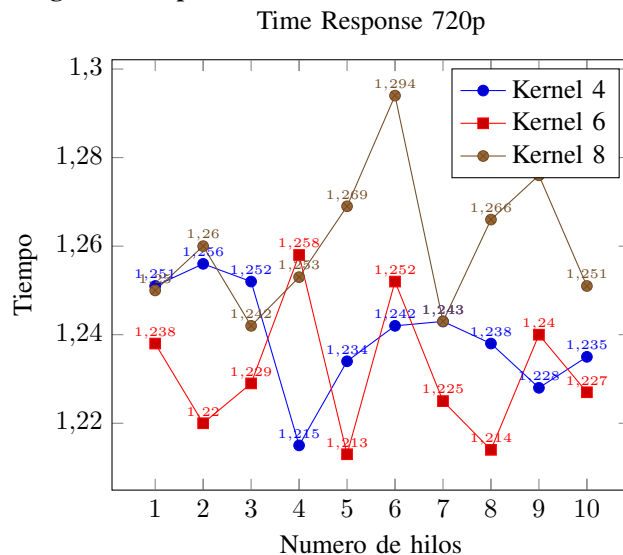
■ Imagen de 4k



III-C. OpenMPI

Para esta practica se utilizo 2 clusters, 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:

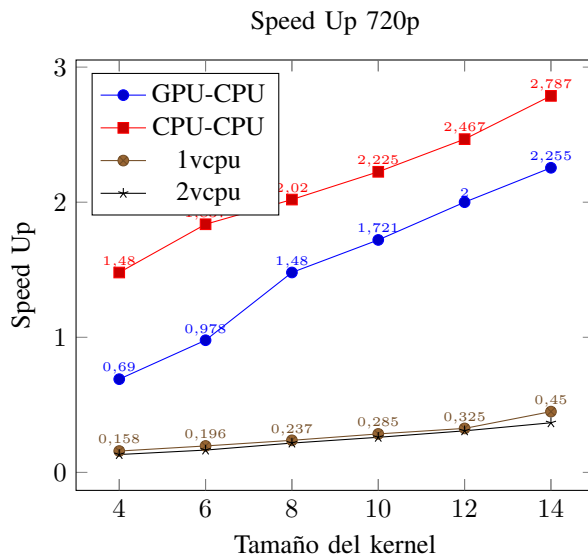
■ Imagen de 720p



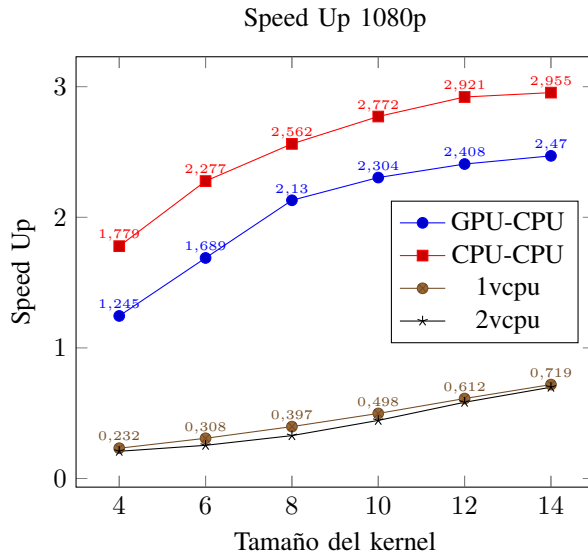
■ Imagen de 1080p

■ Imagen de 720p

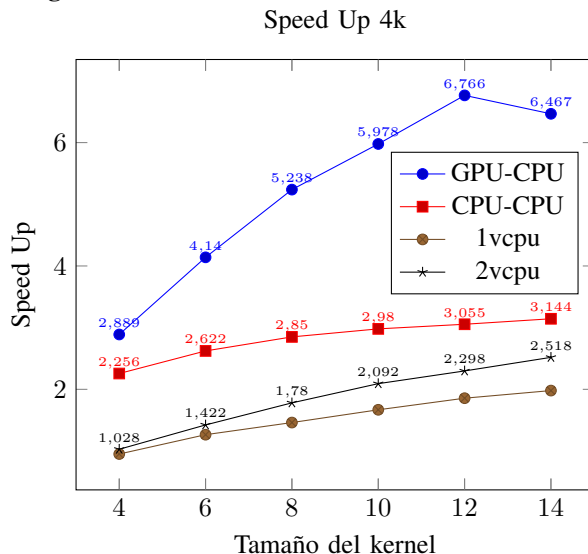
Al momento de graficar el speed up se comparo el de la anterior practica que fue la de CUDA con el speed up de la practica actual. Es decir se hizo la comparación entre GPU-CPU, CPU-CPU, 1 vcpu y 2 vcpu. Las gráficas obtenidas para los diferentes tamaños de imagenes se muestran a continuación:



■ Imagen de 1080p



■ Imagen de 4k



IV. CONCLUSIONES

1. POSIX y OpenMP

- 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 operaciones 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 considerablemente 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*. Available at : [https :
//en.wikipedia.org/wiki/Box_blur](https://en.wikipedia.org/wiki/Box_blur).