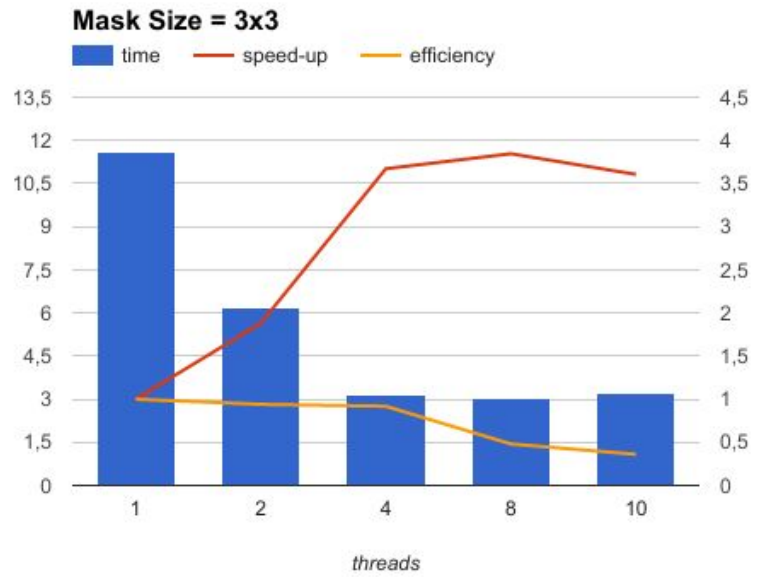
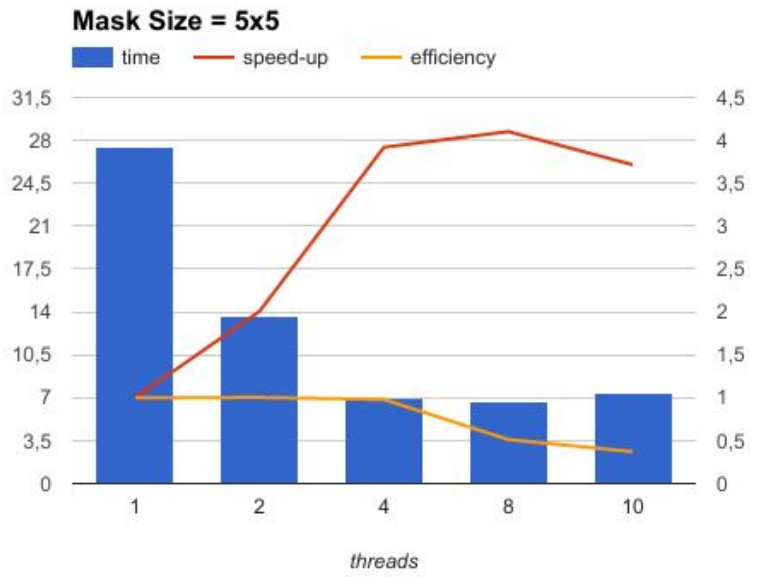
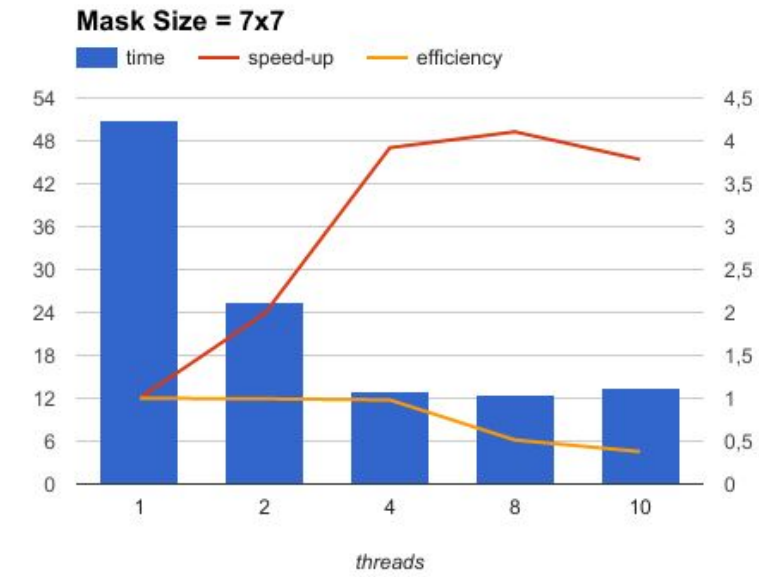
Tarea 1 - HPC - Vicente Dragicevic

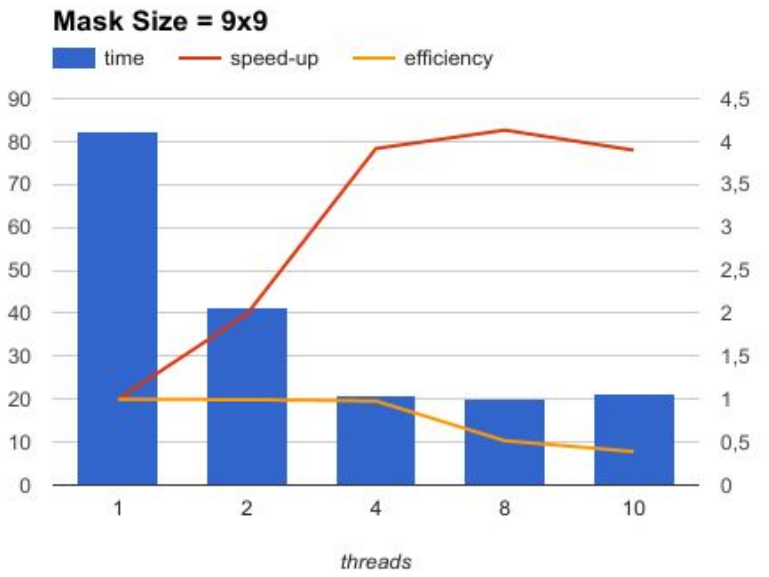
Probé con diferentes tamaños de máscaras (3x3, 5x5, 7x7, 9x9) y threads (1, 2, 4, 8, 10), para más detalles revisar “results.csv” en el directorio de la tarea. Estos resultados fueron generados al correr las pruebas incluidas ($ python3 tests.py). La imagen utilizada para las pruebas se encuentra en el directorio (lake.png, 2048 × 1365 pixeles).

En los siguientes gráficos, el eje de la izquierda corresponde al tiempo en segundos, el eje de la derecha corresponde a *speed-up* y *efficiency*, y el eje de abajo corresponde al número de threads.









Análisis

Aclaración: Sólo mido el tiempo total de aplicar la máscara a toda la imagen (con 10 iteraciones). No tomo en cuenta el tiempo que se demora el programa en leer/escribir las imágenes.

Como podemos observar en los gráficos, prácticamente lo único que cambia es la escala del eje vertical izquierdo (tiempo). Es decir, al aumentar el tamaño de la máscara el tiempo que toma el programa es mayor, pero el speed-up y la eficiencia se mantienen casi constantes. Esto se debe a que el algoritmo que utilizo no paraleliza el cómputo de las máscaras en cada pixel, sino que paraleliza el cómputo del nuevo valor de cada pixel (o sea, cada thread aplica la máscara en su pixel secuencialmente). Decidí paralelizar de esta forma ya que se crean muchos menos threads de los que se crearían al paralelizar el aplicar la máscara.

Se observa una eficiencia “casi perfecta” cuando el programa utiliza 2 y 4 threads, pero ésta disminuye al aumentar a 8 threads. Si bien tripio tiene 8 cores, no necesariamente permitirá que las instrucciones del programa sean ejecutadas en todos los cores. El algoritmo utilizado es *embarrassingly parallel*, por lo que si realmente se estuvieran utilizando los 8 cores la eficiencia debería mantenerse más o menos constante. Una posible optimización sería tomar en cuenta la localidad de los datos, asignándole a cada CPU sectores de la imagen que sean contiguos en memoria y que sean utilizados en los cálculos posteriores.

Independientemente del tamaño de las máscaras, cuando el programa utiliza 10 threads sufre una disminución del speed-up con respecto a cuando utiliza 8 threads. Como sabemos, esto se debe a que al utilizar más threads que cores disponibles se tiene un overhead, debido a que los threads deben compartir los cores (crear/suspender/reanudar threads puede ser caro). Esto además podría producir que no se aproveche tanto el caché, ya que diferentes threads deberán leer distintos datos, y como alguna CPU va a ser compartida por más de un thread su caché podría variar mucho.