



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC3524 — HPC
Tarea OpenMP,
Francesca Lucchini

1 Intrucción

Para esta tarea, la misión era comparar la versión serializada y paraleliada de un código que aplica filtros sobre imágenes. A modo de contexto, el programa utilizado, para aplicar el filtro n veces tiene:

1. Un bloque de *for* que aplica el filtro n veces
2. Un bloque de *for* doble que itera sobre los pixeles de la imagen
3. Un bloque de *for* doble que itera sobre la matriz del filtro y lo aplica para cada pixel.

La versión paralelizada se creó usando directivas de OpenMP sobre el *for* exterior del bloque que itera sobre los pixeles, o sea, tratamos de paralelizar el proceso para cada pixel.

2 Resultados

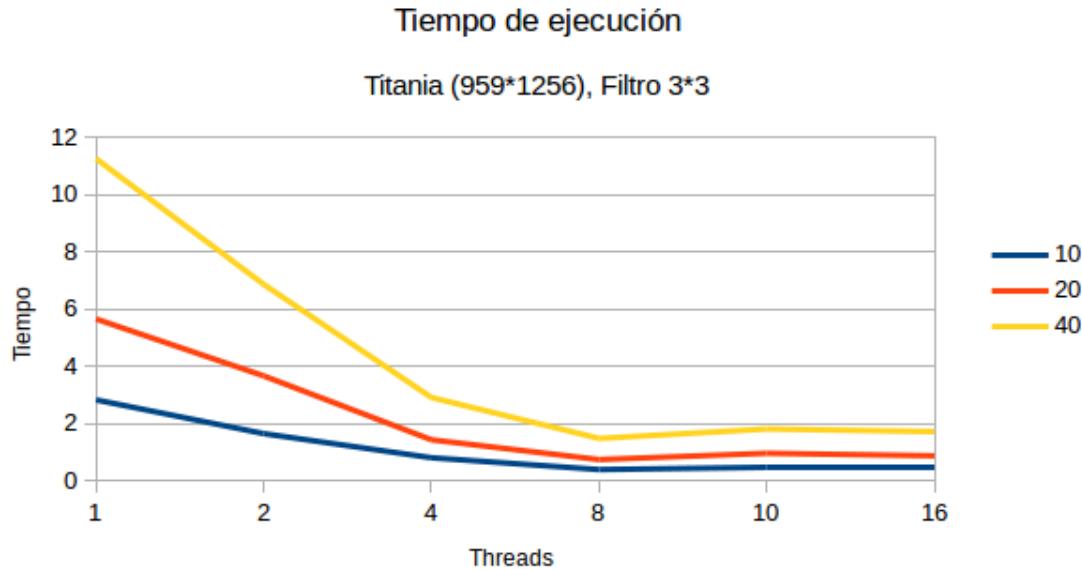
Los test utilizados prueban diferentes números de threads, repeticiones del filtro, tamaños de filtro y tamaños de imágenes. Se generaron 6 scripts que prueban para una imagen y filtro un conjunto $I = 10, 20, 40$ de iteraciones y un conjunto $T = 1, 2, 4, 8, 10, 16$ threads. En concreto:

- Test1: $m3 * 3_i959 * 1256$, aplica un filtro de blur de $3 * 3$ a una imagen de $959 * 1256$ pixeles.
- Test2: $m9 * 9_i959 * 1256$, aplica un filtro de blur de $9 * 9$ a una imagen de $959 * 1256$ pixeles.
- Test3: $m27 * 27_i959 * 1256$, aplica un filtro de blur de $27 * 27$ a una imagen de $959 * 1256$ pixeles.
- Test4: $m3 * 3_i2048 * 1024$, aplica un filtro de blur de $3 * 3$ a una imagen de $2048 * 1024$ pixeles.
- Test5: $m3 * 3_i2782 * 1520$, aplica un filtro de blur de $3 * 3$ a una imagen de $2782 * 1520$ pixeles.
- Test6: $m3 * 3_i4635 * 3051$, aplica un filtro de blur de $3 * 3$ a una imagen de $4635 * 3051$ pixeles.

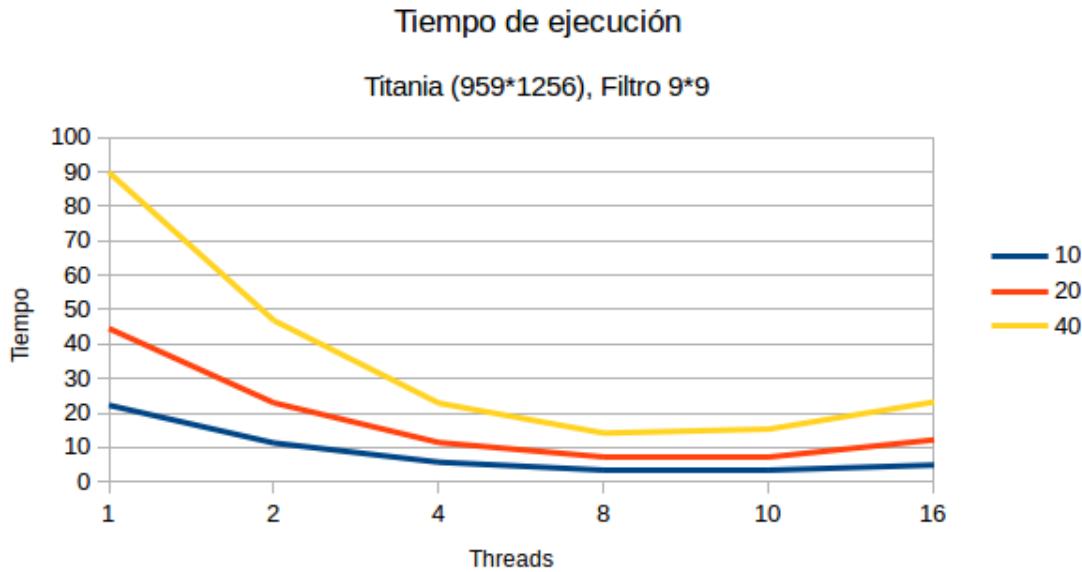
Estos tests se corrieron en Caleuche (1,2,3) y Trauco (4,5,6). Cada color representa un número de repeticiones de un filtro.

2.1 Tiempo de ejecución

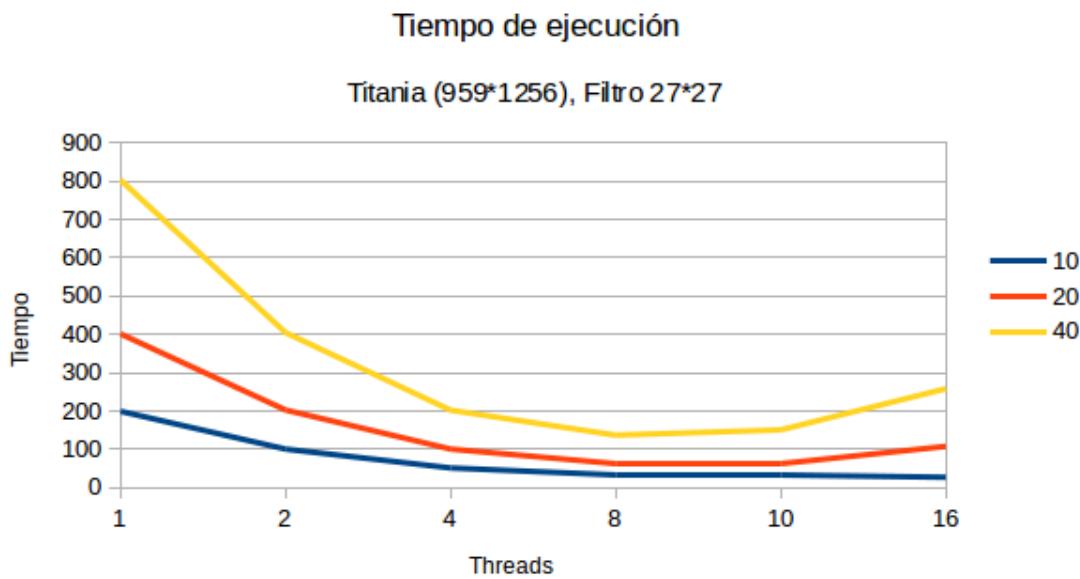
1. Fig 1.1: Test 1



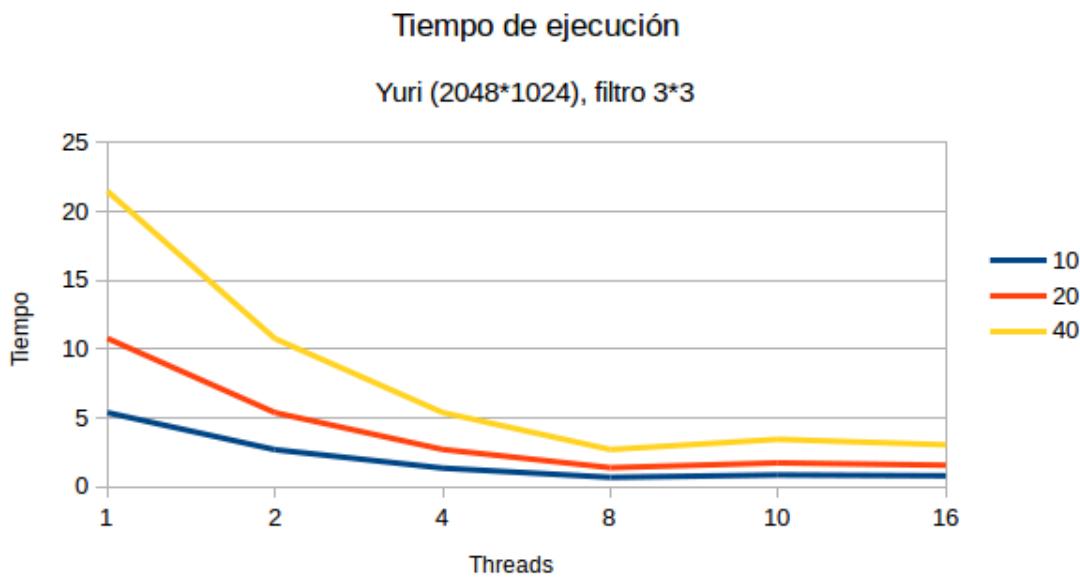
2. Fig 1.2: Test 2



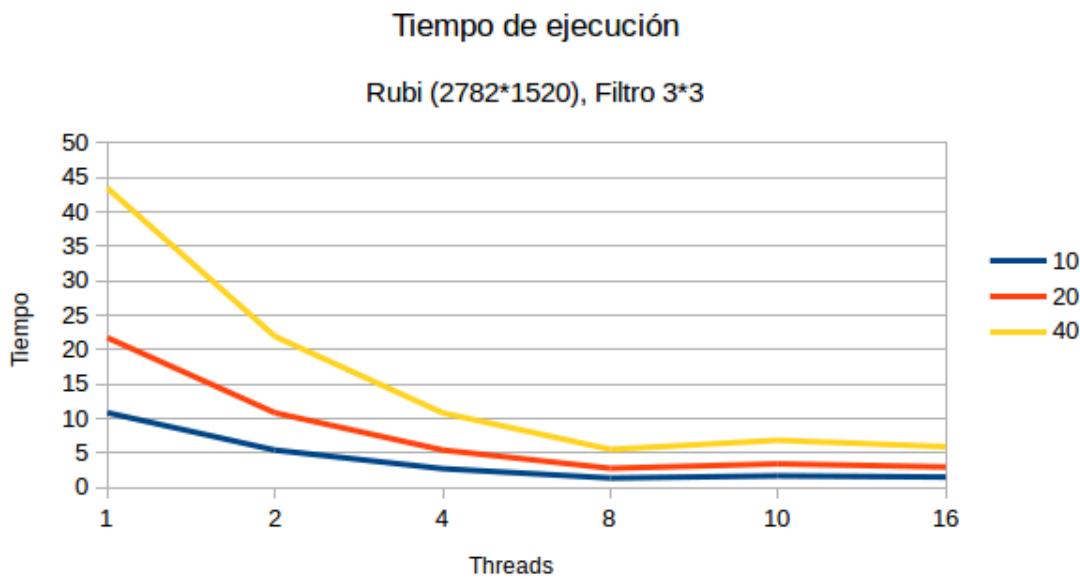
3. Fig 1.3: Test 3



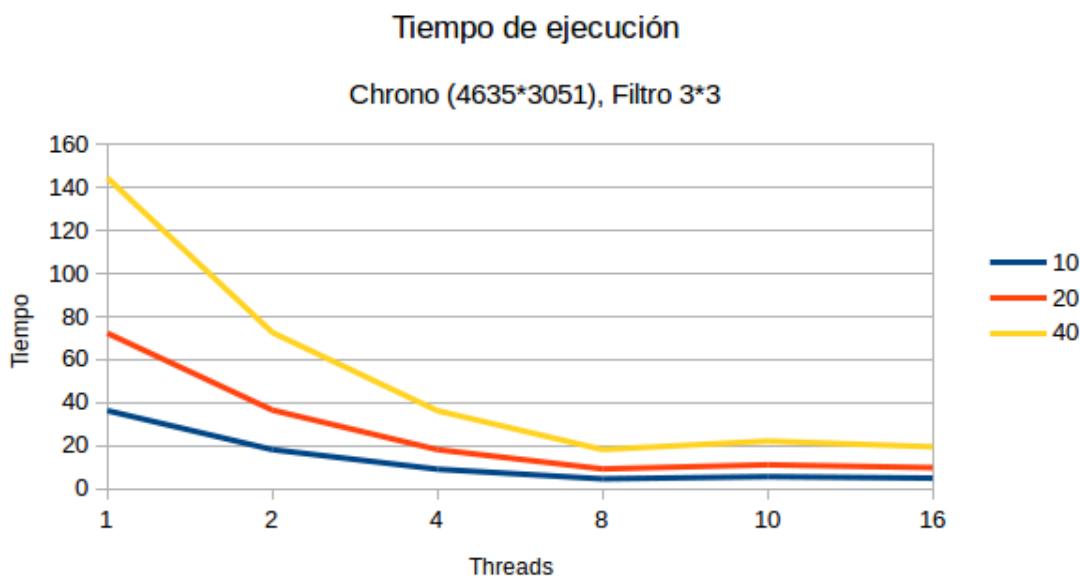
4. Fig 1.4: Test 4



5. Fig 1.5: Test 5

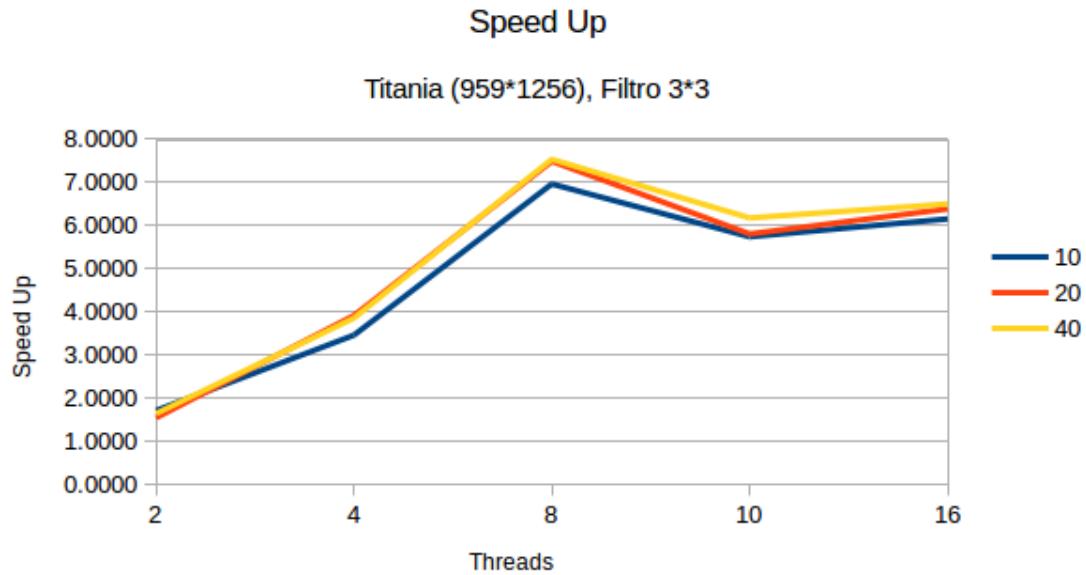


6. Fig 1.6: Test 6

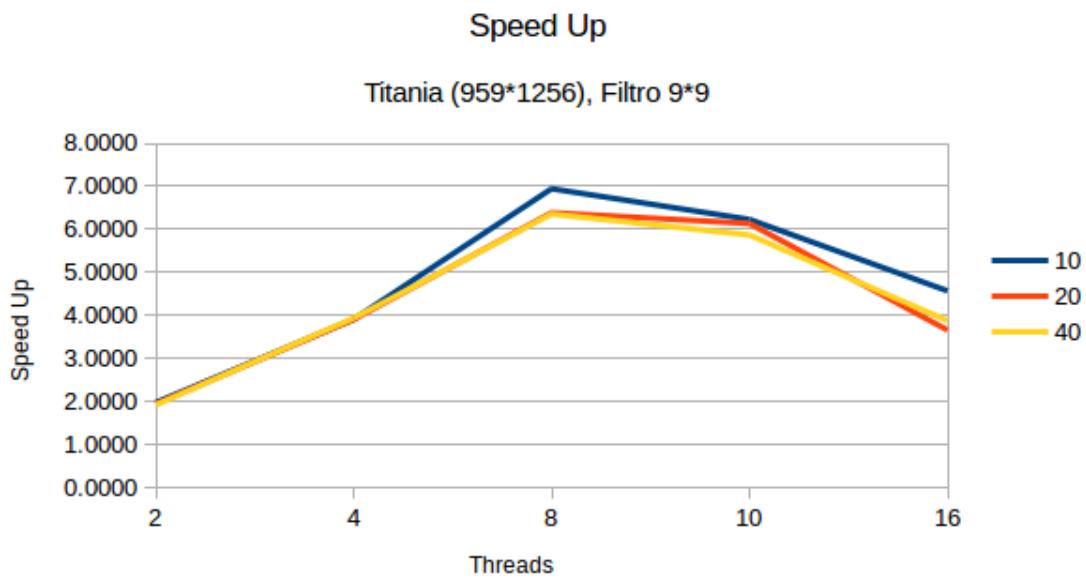


$$2.2 \quad \frac{1}{SpeedUp}$$

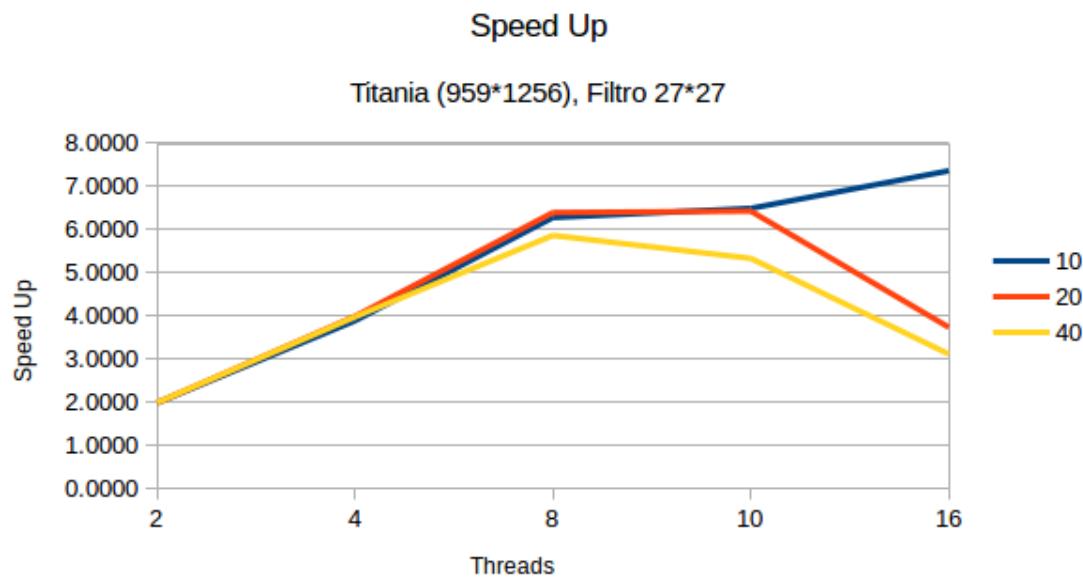
1. Fig 2.1: Test 1



2. Fig 2.2: Test 2



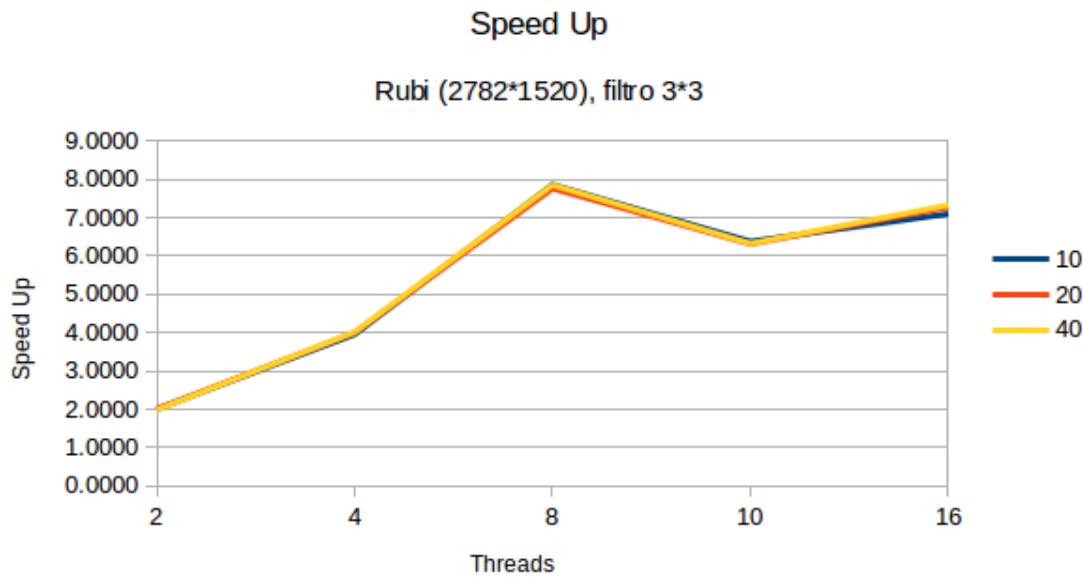
3. Fig 2.3: Test 3



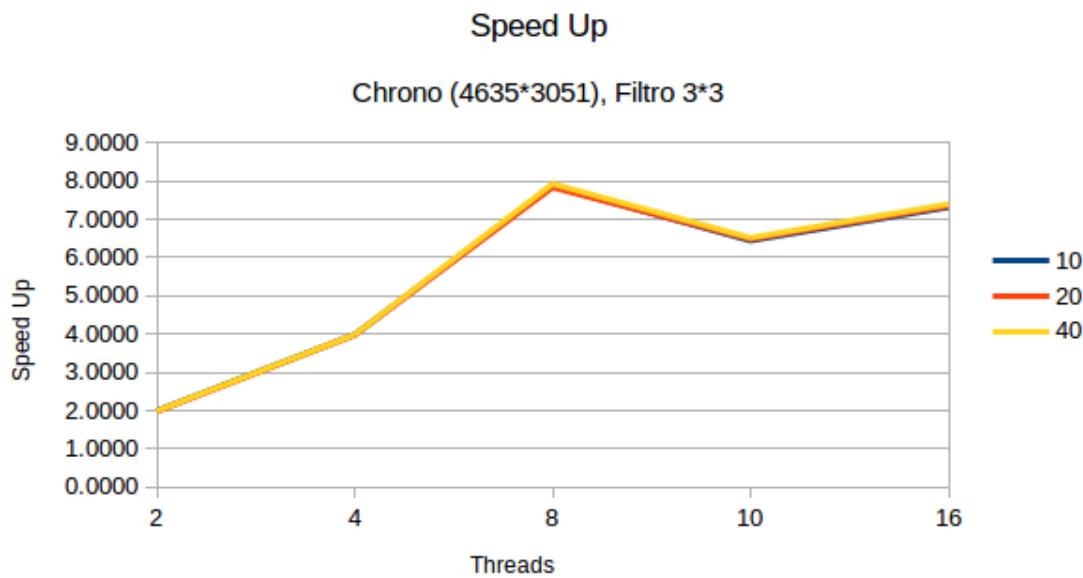
4. Fig 2.4: Test 4



5. Fig 2.5: Test 5

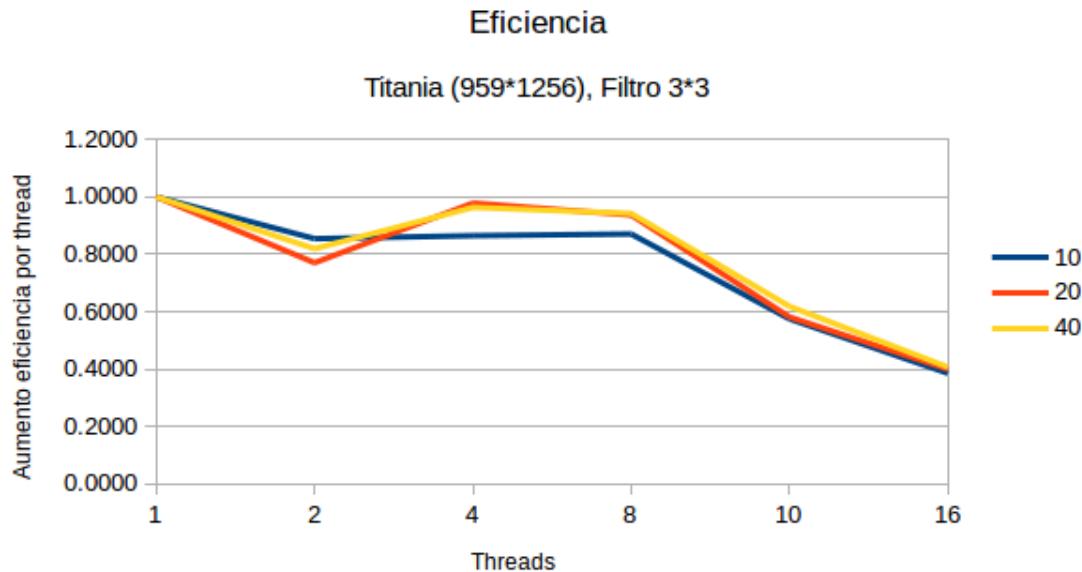


6. Fig 2.6: Test 6

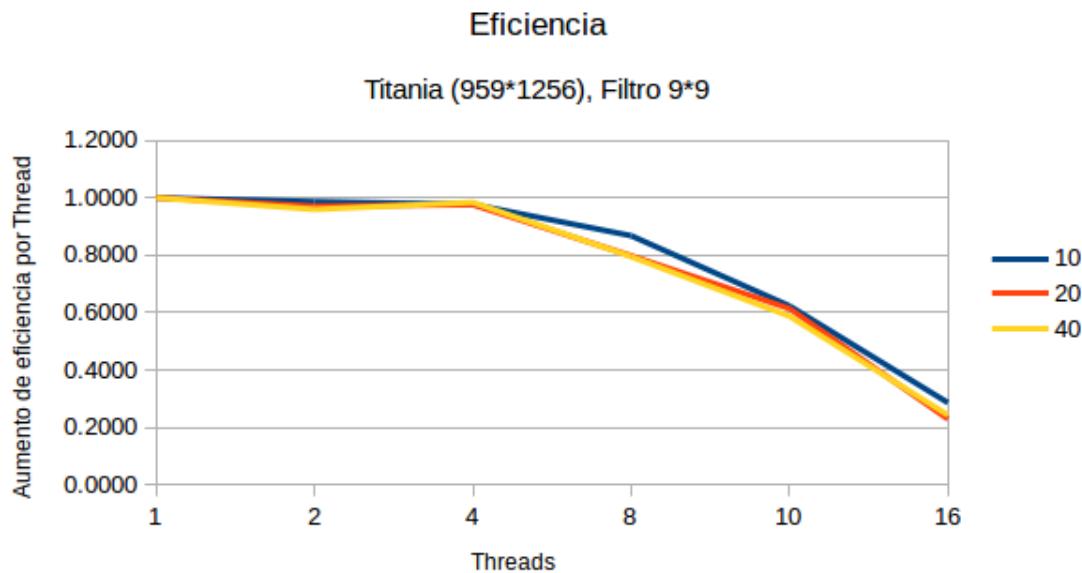


2.3 Eficiencia

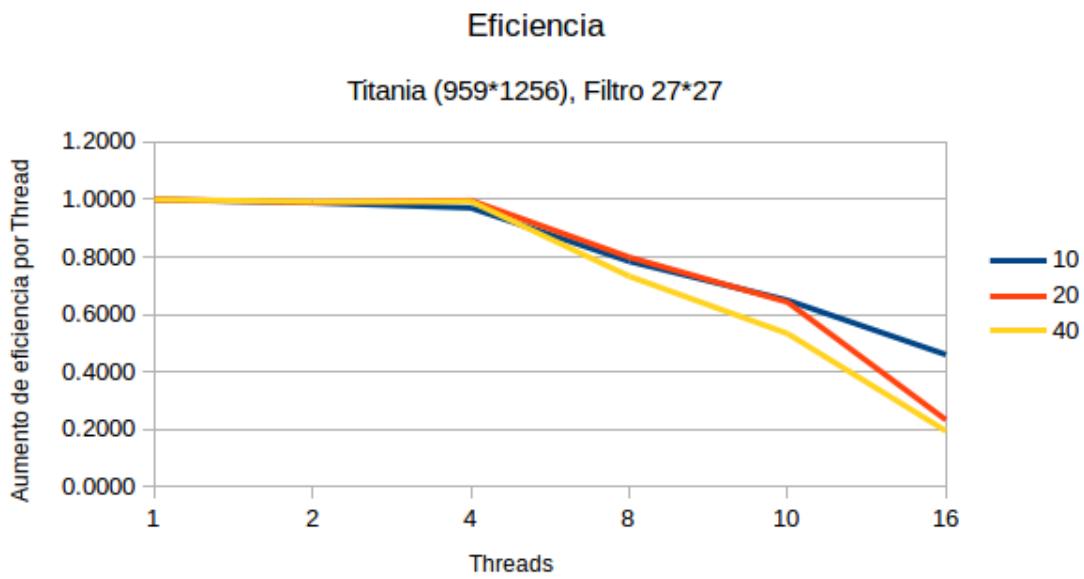
1. Fig 3.1: Test 1



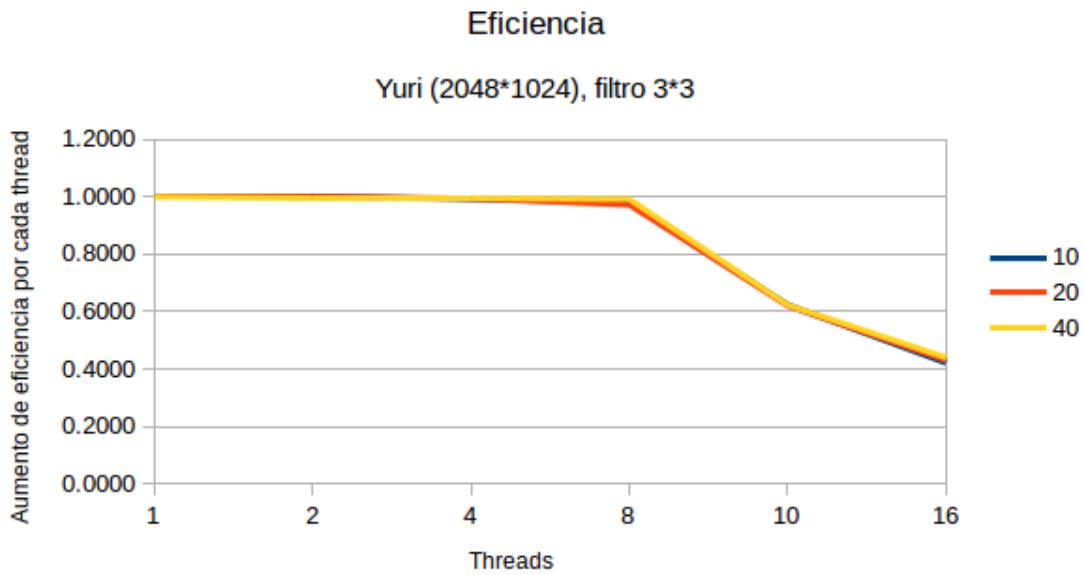
2. Fig 3.2: Test 2



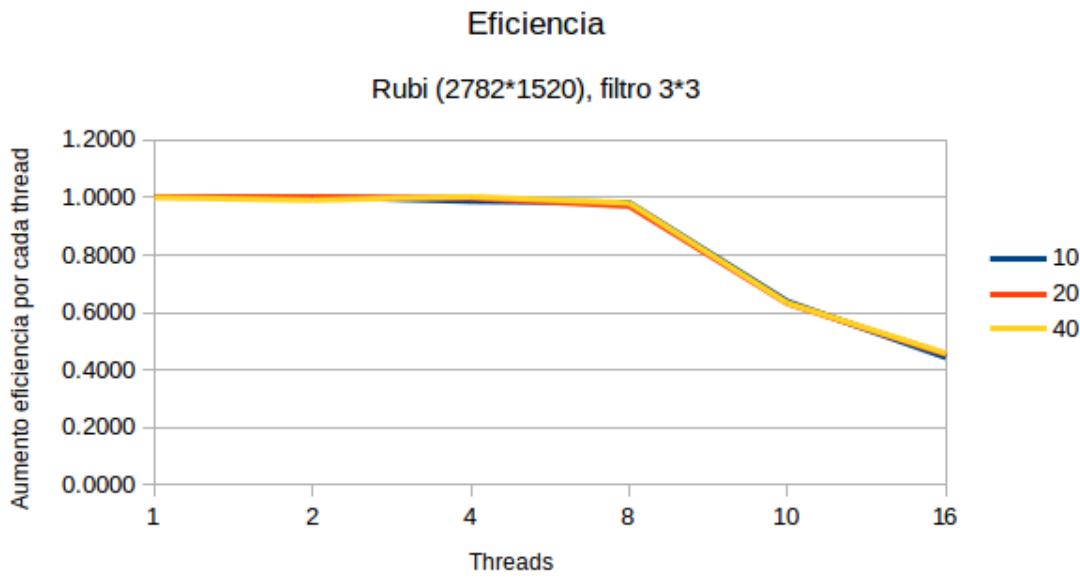
3. Fig 3.3: Test 3



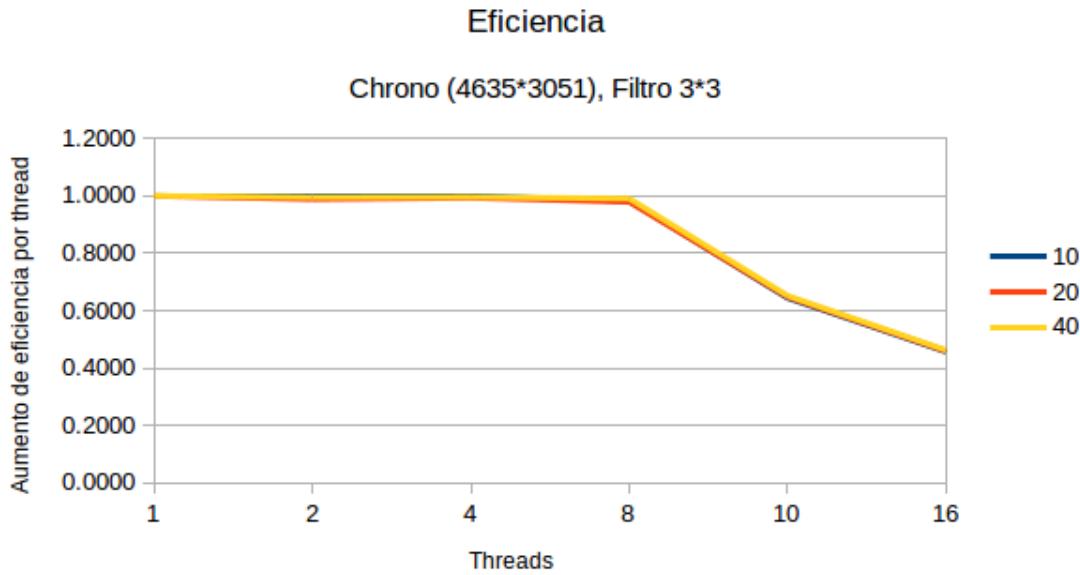
4. Fig 3.4: Test 4



5. Fig 3.5: Test 5



6. Fig 3.6: Test 6



3 Análisis

3.1 Tiempo de ejecución

Al observar Fig 1.1, 1.2 y 1.3, podemos notar que, dada una misma imagen, el aumentar el tamaño del filtro, el tiempo de ejecución aumenta aproximadamente 9 o 10 veces al triplicar el rango de la matriz del filtro. Lo que es esperable, dado que se ejecutan más operaciones por cada pixel y el aplicar el filtro a un pixel no está paralelizado. También podemos notar que, a medida que aumentan los threads, en los 3 gráficos se

genera una curva de estructura similar. Por otro lado, se puede ver que el óptimo número de threads para disminuir el tiempo de ejecución es cercano al número de cores que tiene la máquina (8 en este caso) en el test 2 y 3. Se puede observar un comportamiento similar en Fig 1.4, 1.5 y 1.6.

Algo interesante ocurre cuando el número de threads es mayor al número de cores. Al tener 10 y 16 threads, el la Fig. 1.1, 1.4, 1.5 y 1.6 aumenta el tiempo de ejecución en 10 y en 16 disminuye. Mientras que en la 1.2 y la 1.3, el tiempo de ejecución en 16 threads es mayor al de 10 threads. El primer caso se da cuando el filtro es de rango 3 y el segundo se da cuando el rango de la matriz de filtro es mayor.

3.2 *SpeedUp*

Al observar las figuras 2.1, 2.4, 2.5 y 2.6, podemos notar que el comportamiento del *SpeedUp* es extremadamente similar en las tres curvas de cada gráfico, lo que puede sugerir que el número de repeticiones en las que se aplica un filtro no afecta este parámetro, en el caso de tener un filtro de 3×3 , que es un filtro pequeño. Es interesante notar que después de los 8 threads, el tiempo de ejecución parece acotarse por una constante. Por otro lado, al observar 2.2 y 2.3, la curva azul (10 repeticiones), en vez de mostrar un declive, el *SpeedUp* aumenta al tener más threads que número de cores.

3.3 Eficiencia

Al observar Fig. 3.4, 3.5 y 3.6, podemos notar que, a pesar de que son imágenes con distintas resoluciones, la estructura de las curvas es casi idéntica. También se puede observar que la eficiencia se mantiene casi constante entre 1 a 4 threads. Esto incita a pensar que el aumento a 2 y 4 cores aumenta linealmente el *SpeedUp*, algo que se puede corroborar en la figura 2.4, 2.5 y 2.6. Por otro lado, considerando el aumento de el rango de la matriz del filtro, podemos ver que con un filtro más grande, se puede ver una curva de descenso más pronunciada. En el caso del filtro de tamaño 27 y 10 repeticiones, la eficiencia de cada thread es más alta que en otros casos.

3.4 Conclusiones

Con la información anterior, podemos deducir los siguientes puntos: (i) que el número óptimo de threads para cualquier tipo de imagen es cerca del número de threads de la máquina para casos en el que el filtro es muy grande. Esto puede ser causado porque tener muchos threads puede aumentar el tiempo de ejecución, ya que eso significa que hay más cambios de contexto.

(ii) Al tener una matriz de rango pequeño y que el número de threads supere el número de cores, se observa el tiempo de ejecución sigue disminuyendo. Esto se puede atribuir a que, como lo que debe ejecutar cada thread es muy pequeño, puede resultar más rápido tener más threads en la cola de espera de la CPU, algo que no se da en los casos con un filtro muy grande, ya que el cambio de contexto puede ser más costoso.

(iii) La resolución de la imagen, en el caso de filtro de 3×3 , no afecta la estructura de las curvas de eficiencia. Dado esto, podemos concluir que en filtros pequeños, se mantiene constante para cierto rango de threads y luego disminuye en una curva cóncava y decreciente. Mientras que, con filtros mayores, esta curva se vuelve cónvexa. Eso significa que a medida que el filtro es más grande, la eficiencia que aporta cada thread es menor. Esto se debe probablemente a la razón mencionada anteriormente: el cambio de contexto es más costoso a medida que el kernel es mayor.

4 Imágenes Utilizadas



Figure 1: Usado en test 1, 2 y 3



Figure 2: Usado en test 4



Figure 3: Usado en test 5



Figure 4: Usado en test 6