Modificación Concurrente de imágenes BMP

Primer Proyecto de Sistemas Operativos

Alejandro Uscátegui Torres - CC.1090495286

María José Gómez García - C.C 1000706814

Profesor: Osberth de Castro

Pontificia Universidad Javeriana

Bogotá

2023

Contenido

Pruebas de Rendimiento

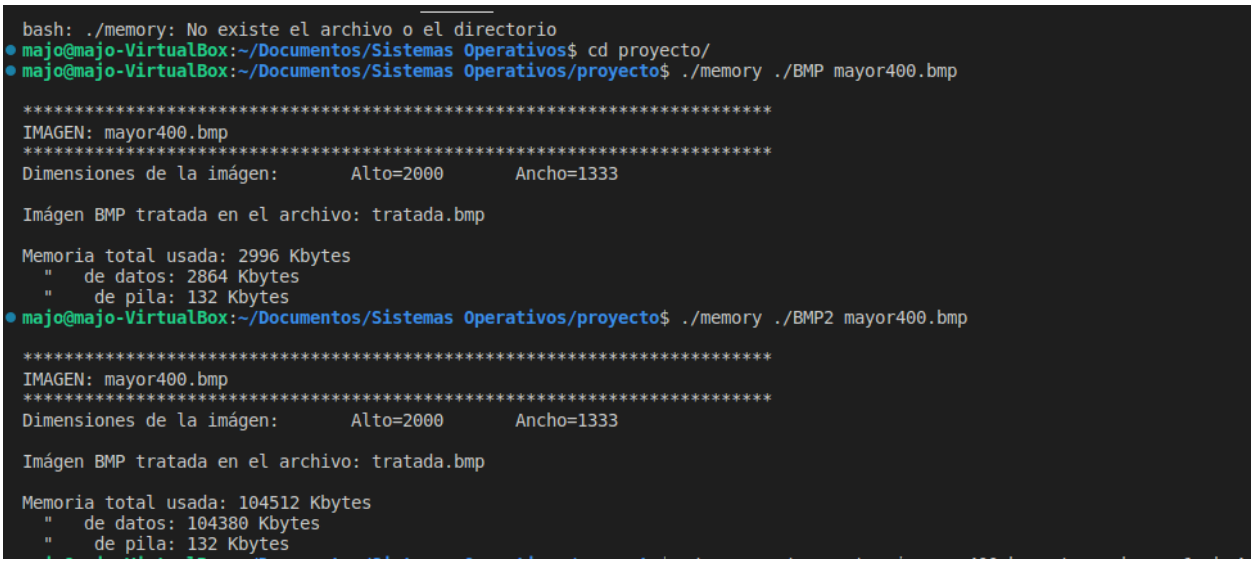
Hpótesis Inicial

1. La hipótesis inicial es que las diferencias en este programa son prácticamente imposibles de notar sin hacer uso de librerías externas porque se ejecutan demasiado rápido, sin embargo, por la cantidad de operaciones y asignaciones es factible asumir que:
    - El programa con las multiplicaciones y las sumas es el más lento porque tiene más operaciones involucradas.
    - El programa de los promedios va a ser el segundo más rápido porque involucra 1 operación menos que el programa de las multiplicaciones, sin embargo, la diferencia será muy baja.
    - Finalmente, el programa de los hilos será el más rápido, pero con base a la cantidad de hilos, creo que si los hilos son muy pocos el programa es más rápido, pero no notablemente más rápido, también creo que si los hilos son muchos, el programa es ineficiente en cuanto debe crear y crear operaciones, a pesar de que funcionen en paralelo, y creo que con la cantidad adecuada de hilos el programa es notablemente más rápido que los otros dos.

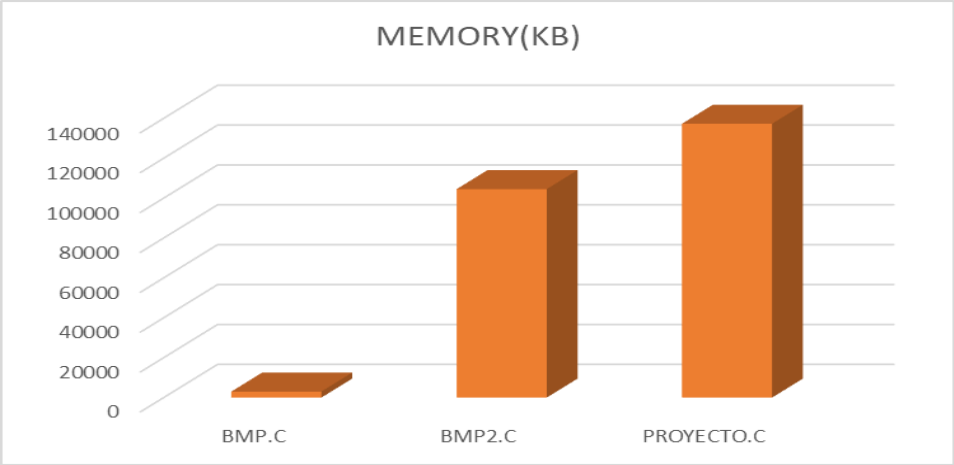
Experimento 1

Se comparará el desempeño de las dos soluciones secuenciales (contenidas en BMP.c y BMP2.c) versus la solución concurrente.

La primera métrica que usaremos será la memoria que ocupan los diferentes programas:



Análisis de datos:



PROGRAMA	MEMORY(KB)
BMP.C	2996
BMP2.C	104512
PROYECTO.C	137280

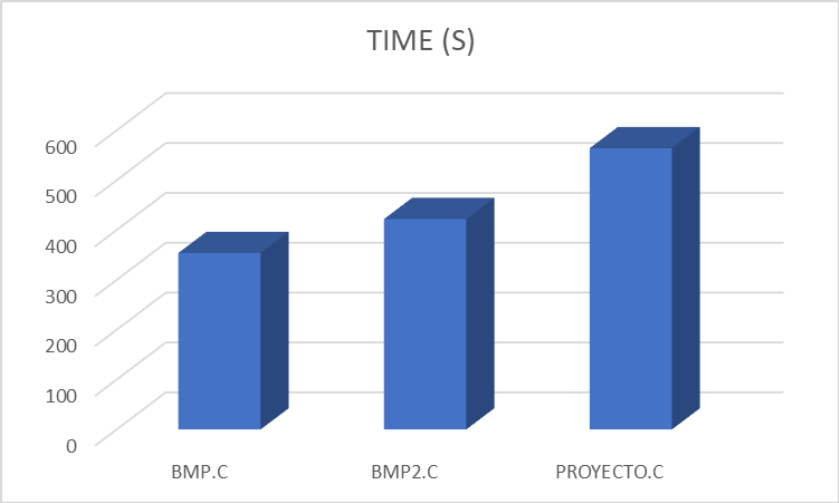
La segunda métrica para comparar los programas es con la función time que estima el tiempo que se demora en correr el programa:

```
majo@majo-VirtualBox:~/Documentos/Sistemas Operativos/proyecto$ time ./BMP mayor400.bmp

**********************************************************************
IMAGEN: mayor400.bmp
**********************************************************************
Dimensiones de la imágen:       Alto=2000       Ancho=1333

Imágen BMP tratada en el archivo: tratada.bmp

real    0m0,354s
user    0m0,254s
sys     0m0,034s
majo@majo-VirtualBox:~/Documentos/Sistemas Operativos/proyecto$
```

```
majo@majo-VirtualBox:~/Documentos/Sistemas Operativos/proyecto$ time ./BMP2 mayor400.bmp

**********************************************************************
IMAGEN: mayor400.bmp
**********************************************************************
Dimensiones de la imágen:       Alto=2000       Ancho=1333

Imágen BMP tratada en el archivo: tratada.bmp

real    0m0,422s
user    0m0,321s
sys     0m0,057s
majo@majo-VirtualBox:~/Documentos/Sistemas Operativos/proyecto$
```

```
majo@majo-VirtualBox:~/Documentos/Sistemas Operativos/proyecto$ time ./proyecto -i mayor400.bmp -t prueba -o 1 -h 4

Imágen BMP original en el archivo: mayor400.bmp

**********************************************************************
imagenOriginal: mayor400.bmp
**********************************************************************
Dimensiones de la imágen:       Alto=2000       |       Ancho=1333

Imágen BMP tratada en el archivo: prueba.bmp

real    0m0,564s
user    0m0,341s
sys     0m0,066s
```

Análisis de datos:



PROGRAMA	TIME (S)
BMP.C	0,354
BMP2.C	0,422
PROYECTO.C	0,564

Así, nuestra hipótesis parece cierta, ya que como se puede observar, el programa que utiliza hilos ocupa más espacio en memoria.

Igualmente, creemos que el uso de funciones alternativas puede a afectar la cantidad de memoria utilizada, pues, aunque no estemos del todo seguros, al ejecutar el programa BMP2, que cuenta sólo con una función más que el programa BMP, el uso de la memoria se vuelve sustancialmente más alto que sin ese llamado extra.

En cuanto a las hipótesis que teníamos sobre el tiempo, las ejecuciones nos muestran que estábamos equivocados.

En un principio pensamos que el programa que usaba hilos iba a ser mucho más rápido pero los resultados son concluyentes, y el programa de hilos no fue el más rápido.

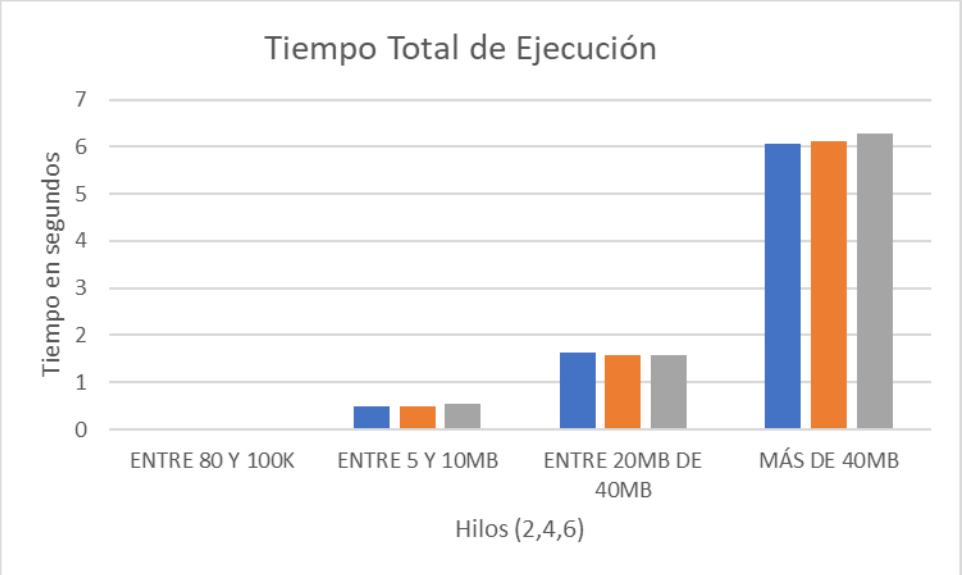
Experimento 2

Estudien el desempeño de la solución concurrente variando el tamaño de la imagen y el número de hilos, tal y como se muestra en la Tabla 1**. Planteen también una hipótesis de trabajo**; realicen las mediciones indicadas en la tabla; grafiquen los resultados (similar a como se muestra en la figura 4) y analicen los resultados obtenidos.

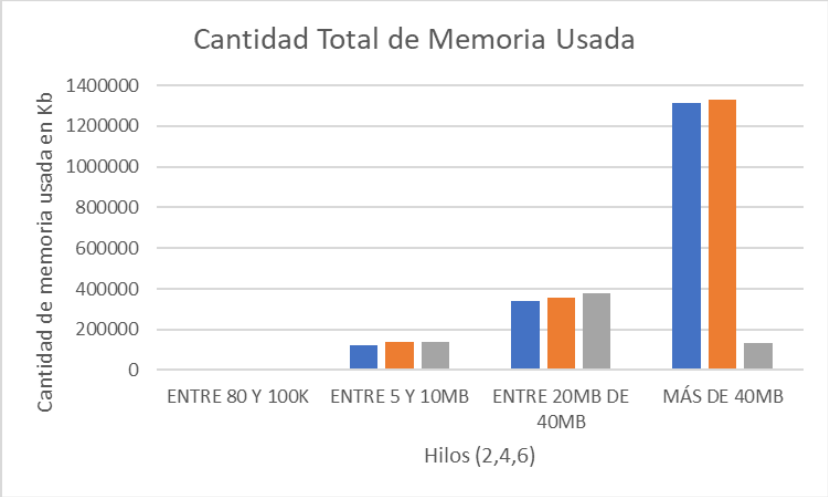
La hipótesis que tenemos para este experimento es que entre mayor sea el peso de la imagen es probable que se tome más tiempo en ejecución porque pensamos que la coordinación y comunicación entre los hilos concurrentes puede resultar en un tiempo de ejecución más lento que una solución secuencial y a la vez ocupe más espacio en memoria. en cuanto a los hilos pensamos que tal vez el uso de múltiples hilos puede

generar sobrecarga en el procesador y la memoria, lo que puede tener un impacto negativo en el rendimiento si no se implementa adecuadamente.

	TIEMPO TOTAL EJECUCIÓN(S)			
NUMERO DE HILOS	ENTRE 80 Y 100K	ENTRE 5 Y 10MB	ENTRE 20MB DE 40MB	MÁS DE 40MB
2	0,019	0,497	1,635	6,062
4	0,018	0,498	1,57	6,125
6	0,019	0,536	1,59	6,279



	MEMORY(KB)			
NUMERO DE HILOS	ENTRE 80 Y 100K	ENTRE 5 Y 10MB	ENTRE 20MB DE 40MB	MÁS DE 40MB
2	16	120896	340716	1312916
4	16	137280	357100	1329300
6	16	137280	376484	134552



En resumen:

1. evidenciamos que a mayor peso de imagen mayor tiempo de ejecución, usará más espacio en memoria.
2. Confirmamos que nuestra inferencia era correcta en cuanto que una cantidad excesiva de hilos afecta negativamente el funcionamiento del programa, así como una cantidad muy baja, sin embargo, esto lo descubrimos al tanteo y no sabemos cómo lograr la solución óptima en cuanto al uso de hilos con modelos teóricos o equivalentes.
3. Descubrimos un evento curioso pues mientras estábamos evaluando la memoria de la imagen con un peso de entre 5 y 10 MB, y usando 4 o 6 hilos el resultado era el mismo. Esto también sucedió con la imagen de un peso entre 80 y 100k MB.

Descripción del filtro creado y la estrategia de división de la imagen

1. Ambas cosas son muy sencillas, experimentando descubrimos que por substracción podíamos generar filtros que tendieran a ser azules, verdes o rojos (RGB) de modo que el filtro creado lo único que hace es eliminar los colores rojos de cada pixel, lo cual genera una imagen muy bonita con nuestra imagen base.

   En segundo lugar, la estrategia de división fue con base al ancho de la imagen, de este modo, dividimos la imagen en cuadrantes, y simplemente la recorríamos hasta que el ancho no diera para más.

   Agregamos de todas maneras una restricción para que el usuario que probara el programa no pudiera poner más hilos que pixeles a lo ancho de la imagen.

2. No hubo demasiado cambios de estilo, pero más bien lo que sí hubo fue una amplia experimentación con programas que nosotros mismos creamos con el objetivo de entender el funcionamiento de los pixeles y hacer filtros chéveres.

Esencialmente, es el mismo programa, porque usa la misma estructura y apertura de imagen y lo que tiene en realidad es un adherido con nuestro nuevo filtro y la funcionalidad por hilos.

3. La plataforma de experimentación fue una computadora con ubuntu linux 22.04, un procesador core i3 de 10ma generación con 4 nucleos, 8 gigabytes de ram y un disco duro de estado sólido de 512 gb. Tomamos la decisión de hacer un dualBoot con un computador directamente porque con el procesador M2 de apple, la máquina virtual presentaba muchas inconsistensias y la versión actual de virtual box, también se estaba buggeando continuamente en nuestro computador con windows y procesador core i5 de 11ava generación.

Conclusión:

Aplicando los conceptos de concurrencia, aprendimos a utilizar diferentes espacios del procesador para hacer óptimo nuestro programa.

Sin embargo, no logramos evidenciar mejorías sustanciales en programas concurrentes y no concurrentes pues los datos nos arrojaron que además de gastar más memoria, los programas concurrentes también gastan más tiempo, esto de todas maneras aclarando que con mediciones que podrían ser injustas pues los programas no realizan exactamente el mismo proceso, aunque sea equivalente.