MEMORIA P4

Daniel Aquino Santiago
Jorge Paniagua Moreno

## Ejercicio 0: información sobre la topología del sistema

```
jorge@jorge-virtual-machine:~/Downloads/materialP4/materialP4$ cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 140
model name      : 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
stepping        : 1
microcode       : 0x60
cpu MHz         : 2803.219
cache size      : 12288 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 27
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov p
liable nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 fma cx16 pcid sse4_1
 ssbd ibrs ibpb stibp ibrs_enhanced fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms
v1 xsaves arat avx512vbmi umip pku ospke avx512_vbmi2 gfni vaes vpclmulqdq avx512
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs itlb_multihit ei
bogomips        : 5606.43
clflush size    : 64
cache_alignment : 64
address sizes   : 45 bits physical, 48 bits virtual
power management:

processor       : 1
vendor_id       : GenuineIntel
cpu family      : 6
model           : 140
model name      : 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
stepping        : 1
microcode       : 0x60
cpu MHz         : 2803.219
cache size      : 12288 KB
physical id     : 2
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 2
initial apicid  : 2
fpu             : yes
fpu_exception   : yes
cpuid level     : 27
wp              : yes
```

En mi caso cuento con una CPU física y una virtual y la opción de hyperthreading no está activa.

## Ejercicio 1: Programas básicos de OpenMP

### 1.1 ¿Se pueden lanzar más threads que cores tenga el sistema? ¿Tiene sentido hacerlo?

Se pueden lanzar más threads que cores, pero no tiene sentido hacerlo, ya que, buscamos que las tareas se realicen en paralelo y no mediante una planificación por parte del SO que alterna en la ejecución de cada uno.

### 1.2 ¿Cuántos threads debería utilizar en los ordenadores del laboratorio? ¿Y en el clúster? ¿Y en su propio equipo?

El número que corresponda al mayor número de cores lógicas que se encuentren en alguno de los procesadores del clúster.

En mi equipo no hay hyperthreading, por lo que se deberán lanzar tantos hilos como cores físicos haya en la cpu, que sería 1 en este caso.

### 1.3 Modifique el programa omp1.c para utilizar las tres formas de elegir el número de threads y deduzca la prioridad entre ellas.

Tras hacer pruebas con las tres formas hemos llegado a la conclusión de que la prioridad de mayor a menor es: cláusula, función y variable de entorno.

EJECUCIÓN omp2

```
jorge@jorge-virtual-machine:~/Downloads/materialP4/materialP4$ ./omp2
Inicio: a = 1,   b = 2,  c = 3
        &a = 0x7ffe830b0274,x   &b = 0x7ffe830b0278,    &c = 0x7ffe830b027c

[Hilo 0]-1: a = -145887856,      b = 2,  c = 3
[Hilo 0]         &a = 0x7ffe830b0210,    &b = 0x7ffe830b0278,    &c = 0x7ffe830b020c
[Hilo 0]-2: a = -1525419761,     b = 4,  c = -1525419773
[Hilo 2]-1: a = 0,       b = 4,  c = 3
[Hilo 2]         &a = 0x7f4cdd3fdd80,    &b = 0x7ffe830b0278,    &c = 0x7f4cdd3fdd7c
[Hilo 2]-2: a = 21,      b = 6,  c = 3
[Hilo 1]-1: a = 0,       b = 6,  c = 3
[Hilo 1]         &a = 0x7f4cddbfed80,    &b = 0x7ffe830b0278,    &c = 0x7f4cddbfed7c
[Hilo 1]-2: a = 27,      b = 8,  c = 3
[Hilo 3]-1: a = 0,       b = 8,  c = 3
[Hilo 3]         &a = 0x7f4cdcbfcd80,    &b = 0x7ffe830b0278,    &c = 0x7f4cdcbfcd7c
[Hilo 3]-2: a = 33,      b = 10,         c = 3

Fin: a = 1,      b = 10,         c = 3
        &a = 0x7ffe830b0274,    &b = 0x7ffe830b0278,    &c = 0x7ffe830b027c
```

### 1.4 ¿Cómo se comporta OpenMP cuando declaramos una variable privada?

OpenMP crea una nueva variable con el mismo nombre de la privada. La nueva variable tiene una dirección de memoria distinta, por lo que al sufrir modificaciones, estas no se reflejan en la variable del thread master. Cada thread tiene una dirección de memoria distinta para esta variable.

## 1.5 ¿Qué ocurre con el valor de una variable privada al comenzar a ejecutarse la región paralela?

Si la variable está definida como firstprivate, su valor comienza siendo el mismo que su valor en la región no paralela. Si la variable está definida como private su valor inicial no coincide con el de la región no paralela. Tendrá el valor que tenga la región de memoria que se le asigna.

## 1.6 ¿Qué ocurre con el valor de una variable privada al finalizar la región paralela?

En este caso mantiene el valor que tenía antes de entrar en la región paralela.

## 1.7 ¿Ocurre lo mismo con las variables públicas?

No, viendo el ejemplo la dirección de memoria de la variable pública nos damos cuenta que es la misma tanto en ambas regiones, por lo que cuando el valor se modifica en la región paralela también se ve modificada el valor en la región no paralela.

## Ejercicio 2: Paralelizar el producto escalar

### 2.1 Ejecute la versión serie y entienda cuál debe ser el resultado para diferentes tamaños de vector.

El programa realiza el producto escalar de dos vectores, por lo que el resultado siempre será el tamaño de los vectores.

### 2.2 Ejecute el código paralelizado con el pragma openmp y conteste en la memoria a las siguientes preguntas:

**• ¿Es correcto el resultado?**

```
jorge@jorge-virtual-machine:~/Downloads/materialP4/materialP4$ ./pescalar_serie
Resultado: 1000000.000000
Tiempo: 0.014302
jorge@jorge-virtual-machine:~/Downloads/materialP4/materialP4$ ./pescalar_par1
Se han lanzado 2 hilos.
Resultado: 834992.000000
Tiempo: 0.011332
```

El resultado no es correcto, debería salir el mismo que el de la ejecución en serie.

**• ¿Qué puede estar pasando?**

La variable sum se establece como shared por defecto en openmp. Esto causa que todos los hilos lean y escriban de la misma variable, por lo cual los hilos muchas veces no están leyendo el valor real de la variable.

### 2.3 Modifique el código anterior y denomine el programa pescalar_par2.

```
jorge@jorge-virtual-machine:~/Downloads/materialP4/materialP4$ ./pescalar_serie
Resultado: 1000000.000000
Tiempo: 0.008054
jorge@jorge-virtual-machine:~/Downloads/materialP4/materialP4$ ./pescalar_par2
Se han lanzado 2 hilos.
Resultado: 1000000.000000
Tiempo: 0.044011
```

**• ¿Puede resolverse con ambas directivas? Indique las modificaciones realizadas en cada caso.**

Sí, en el caso de la directiva atomic debemos reescribir la sentencia de sum=sum+A[k]*B[k] a sum+=A[k]*B[k]. Para la directiva critical únicamente hay que meter el código entre {}.

**• ¿Cuál es la opción elegida y por qué?**

La opción elegida es: #pragma omp atomic, ya que, se trata de una única sentencia RMW sencilla y tras hacer algunas pruebas se ha observado que es más rápida.

### 2.4 Modifique el código anterior y denomine el programa resultante pescalar_par3.

```
jorge@jorge-virtual-machine:~/Downloads/materialP4/materialP4$ ./pescalar_par3 20 2
Se han lanzado 2 hilos.
Resultado: 20.000000
Tiempo: 0.000020
jorge@jorge-virtual-machine:~/Downloads/materialP4/materialP4$ ./pescalar_par3 50 2
Se han lanzado 2 hilos.
Resultado: 50.000000
Tiempo: 0.000012
```

**• Comparando con el punto anterior ¿Cuál será la opción elegida y por qué?**

La opción elegida será el pragma reduction porque llegamos a un resultado correcto y el tiempo de ejecución muestra diferencias bastante notables, siendo mucho más rápido que las opciones anteriores.

## Ejercicio 3: Paralelizar el producto escalar

Solo cuento con 2 cores por lo que para un número mayor de 2 no tiene sentido.

**TAMAÑO 1000 x 1000:**

| versión/#hilo | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| serie | 8.320 | | | |
| paralela-bucle1 | 5.879 | 3.949 | | |
| paralela-bucle2 | 7.834 | 5.352 | | |
| paralela-bucle3 | 8.714 | 4.460 | | |

**TAMAÑO 1800 x 1800:**

| version/#hilo | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| serie | 58.03 | | | |
| paralela-bucle1 | 46.71 | 27.81 | | |
| paralela-bucle2 | 55.99 | 44.92 | | |
| paralela-bucle3 | 55.68 | 49.15 | | |

### 3.1 ¿Cuál de las tres versiones obtiene peor rendimiento? ¿A qué se debe? ¿Cuál de las tres versiones obtiene mejor rendimiento? ¿A qué se debe?

El programa con peor rendimiento es el que tiene el bucle más externo paralelizado, debido a que se están usando más recursos de los que son necesarios, aumentando así el tiempo de ejecución.

El programa con mejor rendimiento es el que tiene el bucle más interno paralelizado, debido a que hace uso de los recursos estrictamente necesarios.

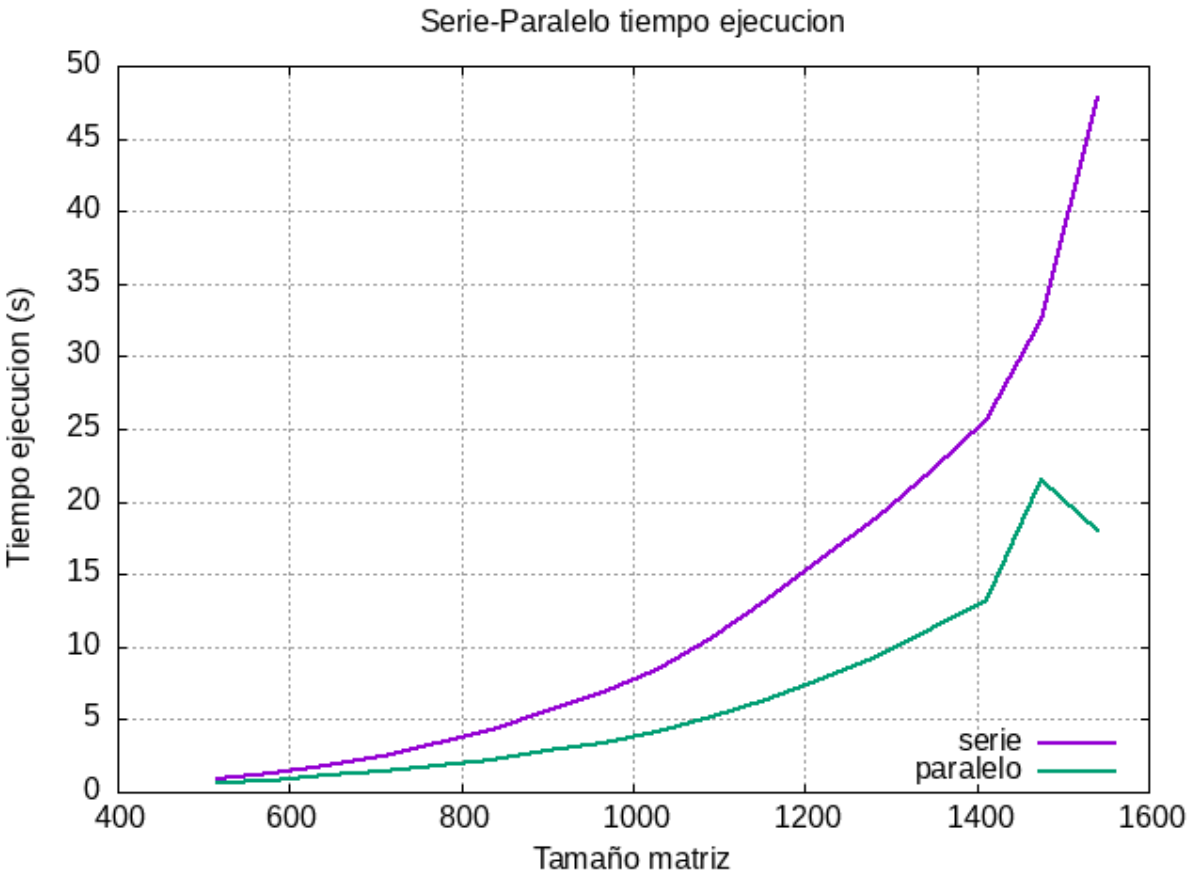
### 3.2 En base a los resultados, ¿cree que es preferible la paralelización de grano fino (bucle más interno) o de grano grueso (bucle más externo) en otros algoritmos?

Viendo los resultados creo que la paralelización de grano fino es preferible en otros algoritmos similares.

### 3.3 Si en la gráfica anterior no obtuvo un comportamiento de la aceleración en función de N que se estabiliza o decrece al incrementar el

**tamaño de la matriz, siga incrementando el valor de N hasta conseguir una gráfica con este comportamiento e indique para qué valor de N se empieza a ver el cambio de tendencia.**

P = 11%7+1 = 4, observamos que para tamaños pequeños las diferencias entre tiempos de ejecución no son significativas. Sin embargo, a partir de un tamaño de aproximadamente 1000 elementos las diferencias que se observan comienzan a tomar importancia.




## Ejercicio 4: Ejemplo de integración numérica

## Pregunta 4.1:

## ¿Cuántos rectángulos se utilizan en la versión del programa que se da para realizar la integración numérica?

El número de rectángulos utilizados para realizar la integración numérica está determinado por la variable n, en este caso su valor es 100000000.

## ¿Cuál es entonces el valor de *h* (ancho del rectángulo)?

El valor de h está determinado por la división de la longitud total del intervalo entre el número de rectángulos(n).

```
h = 1.0/(double) n;
```

## Pregunta 4.2: Ejecuta todas las versiones paralelas y la versión serie. Analice el rendimiento (en términos de tiempo medio de ejecución y la aceleración o speedup) y si el resultado obtenido es correcto o no, reflejando estos datos en una tabla. En las versiones que el resultado no sea correcto, incluya una breve explicación de por qué no es correcto.

En este caso trabajaremos con 6 cores
El speedup, se calculará como S = Toriginal/Toptimizado , siendo “Toriginal” el tiempo de ejecución en serie, y “Toptimizado”, el tiempo de cada programa en paralelo

| Programa | Tiempo de ejecución | SpeedUp |
|---|---|---|
| pi_serie | 0.279838 | X |
| pi_par1 | 0.277862 | 1.007 |
| pi_par2 | 0.291959 | 0.9584 |
| pi_par3 | 0.046036 | 6.0786 |
| pi_par4 | 0.055768 | 5.0179 |
| pi_par5 | 0.046351 | 6.0374 |
| pi_par6 | 0.281627 | 0.9936 |
| pi_par7 | 0.046206 | 6.0563 |

Restultados Incorrectos:

## Ejercicio 4.3: En la versión pi_par2, ¿tiene sentido declarar sum como una variable privada?

En este caso sum se declara dentro de la región paralela, lo cual hace que cada hilo tenga su propia copia local de sum, por lo tanto, agregar “private(sum)” no es necesario.

## ¿Qué sucede cuando se declara una variable de tipo puntero como privada?

Cada hilo tendra su propia copia del puntero, pero no de los datos a los que se apuntan en dicho puntero.
En este caso cada hilo tiene su propia copia de “sum” pero todos los punteros “sum” apuntan al mismo bloque de memoria, esto puede conducir a “False Sharing” si los hilos están escribiendo en ubicaciones de memoria cercanas.

## Ejercicio 4.4: ¿Cuál es la diferencia entre las versiones pi_par5, pi_par3 y la versión pi_par1?

1. “pi_par1” y “pi_par3” utilizan punteros a sum para almacenar las sumas parciales, mientras que “pi_par5” utiliza una única variable “sum” y una región crítica.
2. “pi_par5” utiliza una región crítica para acumular los resultados finales, lo cual puede afectar al rendimiento en comparación con los punteros individuales.
3. “pi_par3” realiza una verificación del tamaño de la línea de cache y ajusta el tamaño del puntero “sum” en consecuencia

## Explique lo que es False sharing e indique qué versiones se ven afectadas por ello (y en qué medida).

Es un fenómeno que ocurre cuando dos o más hilos comparten la misma línea de caché, pero cada hilo está escribiendo en ubicaciones de memoria diferentes dentro de esa línea.

1. “pi_par1” debido al puntero “sum”. En este caso cada hilo opera en una sección independiente del puntero, lo que podría conducir a false sharing.
2. “pi_par5” debido a que utiliza una única variable “sum” en vez de un puntero compartido.
3. “pi_par3” en este caso se incluyen medidas preventivas del false sharing.

## ¿Por qué en pi_par3 se obtiene el tamaño de línea de caché?

Como he introducido previamente este fichero goza de medidas preventivas de false sharing, una de ellas es el uso de padding para ajustar el tamaño del puntero “sum” y reducir el impacto en la caché y maximizar el rendimiento.

## Ejercicio 4.5: Explique el efecto de utilizar la directiva critical. ¿Qué diferencias de rendimiento se aprecian?

Se utiliza para especificar una región crítica en el código en la que solo un hilo puede ejecutar dicha parte de código a la vez.
Evita condiciones de carrera, lo que produce que se actualice en cada hilo uno por uno el valor de “sum”.Esto puede causar efectos positivos, como la consistencia o negativos como bloque de hilos, lo que puede conllevar a un mayor tiempo de ejecución.

## ¿A qué se debe este efecto?

Los hilos se quedan esperando a que otro hilo finalice la operación dentro de la zona crítica.

## Ejercicio 4.6: Ejecuta la versión pi_par6 del programa. ¿Qué diferencias de rendimiento se aprecian? ¿A qué se debe este efecto?

Se puede observar un tiempo de ejecución más lento, que para otros ejemplos. Se genera un array de sum,y mediante un patrón de reducción se calcula el valor de sum de cada hilo

y finalmente se realiza una suma total de las diferentes sumas parciales, esto evita la necesidad de una sección crítica, pero se genera una mayor necesidad de bucles y accesos a memoria.

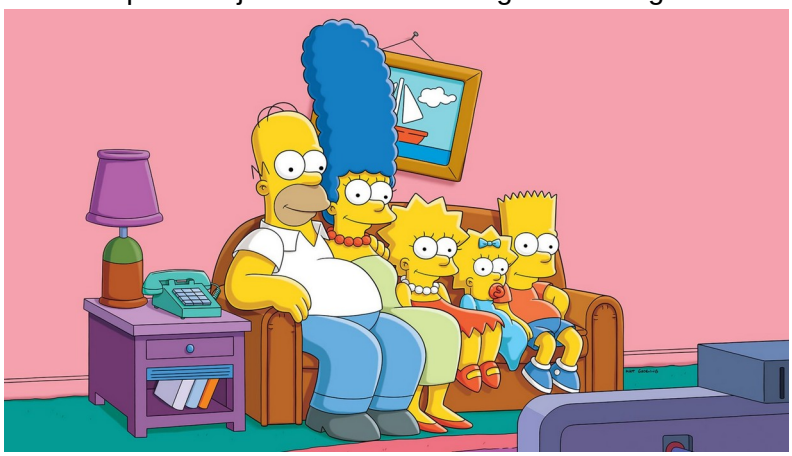
## Ejercicio 4.7: ¿Qué versión es la óptima y por qué?

Depende del contexto.
Por ejemplo, “pi_par7” con la cláusula “reduction” es una opción fuerte, ya que es la más simple y limpia, sin embargo para problemas grandes puede ser beneficioso el uso de “pi_par3” que considera el tamaño de la línea de cache, y utiliza un array con padding para evitar false sharing

## Ejercicio 5: Optimización de programas de cálculo

**0. Compile y ejecute el programa usando como argumento una imagen de su elección. Examine los ficheros que se hayan generado y analice el programa entregado.**

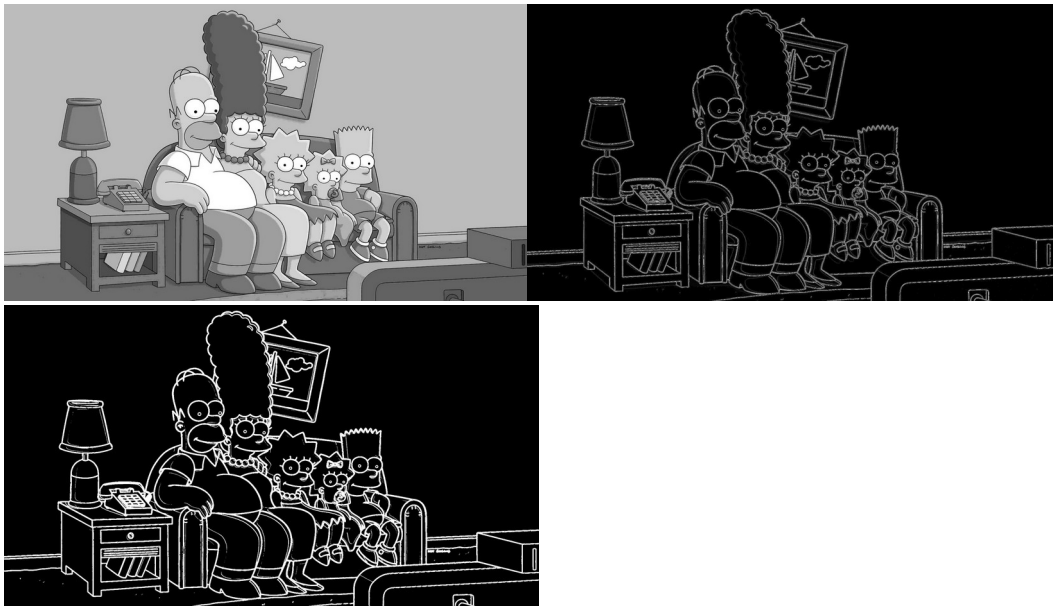
Tras compilar el ejecutable sobre la siguiente imagen



de diferentes formatos: 4k, 8k, hd, fhd y sd
obtenemos el siguiente resultado

```
~/Do/i/t/p/ar/p/P4_1323_P11 ./edgeDetector 4k.jpg
[info] Processing 4k.jpg
[info] 4k: width=3840, height=2160, nchannels=3
[info] Using gaussian denoising...
Tiempo: 1.305916
~/Do/i/t/p/ar/p/P4_1323_P11 ./edgeDetector 8k.jpg
[info] Processing 8k.jpg
[info] 8k: width=7680, height=4320, nchannels=3
[info] Using gaussian denoising...
Tiempo: 6.888474
~/Do/i/t/p/ar/p/P4_1323_P11 ./edgeDetector HD.jpg
[info] Processing HD.jpg
[info] HD: width=1280, height=720, nchannels=3
[info] Using gaussian denoising...
Tiempo: 0.117625
~/Do/i/t/p/ar/p/P4_1323_P11 ./edgeDetector FHD.jpg
[info] Processing FHD.jpg
[info] FHD: width=1920, height=1080, nchannels=3
[info] Using gaussian denoising...
Tiempo: 0.279735
~/Do/i/t/p/ar/p/P4_1323_P11 ./edgeDetector SD.jpg
[info] Processing SD.jpg
[info] SD: width=640, height=360, nchannels=3
[info] Using gaussian denoising...
Tiempo: 0.030363
```

Se generan las siguientes imágenes para 4j.jpg



Primero encontramos la imagen en escala de grises, otra con escala de grises pero con lo ejes diferenciados, y finalmente una imagen en blanco y negro con los ejes claros.
El programa para realizar esto itera sobre todos los pixels de la imagen y aplica una transformación de colores 0.2989R + 0.5870G + 0.1140B
El programa consta de la función “gaussian_kernel” la cual crea el kernel gaussiano, la función getRGB que obtiene los valores RGB de una localización específica (x, y), función de comparación “comparate”.
El main, realiza una serie de operaciones en cada imagen de entrada, conversión de RGB a escala de grises, detección de bordes usando el operador de Sobel, eliminación de ruido, y por último escritura de imágenes de salida.

**1. El programa incluye un bucle más externo que itera sobre los argumentos aplicando los algoritmos a cada uno de los argumentos (señalado como Bucle 0). ¿Es este bucle el óptimo para ser paralelizado?**

No sería el óptimo ya que dependiendo del tamaño de imágenes que se inserten en la entrada, algún proceso puede estar consumiendo gran cantidad de recursos.

**Responda a las siguientes cuestiones para complementar su respuesta.**
**a. ¿Qué sucede si se pasan menos argumentos que número de cores?**

Los núcleos restantes estarán inactivos, mientras los otros trabajan

**b. Suponga que va a procesar imágenes de un telescopio espacial que ocupan hasta 6GB cada una, ¿es la opción adecuada? Comente cuanta memoria consume cada hilo en función del tamaño en pixeles de la imagen de entrada**

Cada hilo va a necesitar reservar memoria para las imágenes y para las imágenes creadas tras realizar las operaciones de procesamiento de imágenes. Como cada imagen ocupa hasta 6GB y una imagen RGB contiene píxeles de 3 byte el número de píxeles que tendrá la imagen es de $2^{31}$ píxeles. Cada hilo reservará memoria para las tres imágenes de salida, cada una de ellas es una array de bytes, uno por cada píxel, por lo que será 6GB de memoria más los 6GB de la imagen de entrada. Cada hilo además necesitará el doble de la memoria de la imagen que está procesando. Finalmente cada hilo libera la memoria al final del programa lo que genera que la memoria utilizada en total sea más grande.

**2. Durante la práctica anterior, observamos que el orden de acceso a los datos es importante. ¿Hay algún bucle que esté accediendo en un orden subóptimo a los datos? Corríjalo en tal caso.**
**a. Es imprescindible que el programa siga realizando el mismo algoritmo, por lo que solo se deberían realizar cambios en el programa que no cambien la salida.**
**b. Explique por qué el orden no es el correcto en caso de cambiarlo.**

Efectivamente, se han modificado una serie de bucles, dichas modificaciones se pueden encontrar en el archivo “edgeDetector-5-2.c”

En la parte “RGB to grey image” podemos encontrar “grey_image[j * width + i]”, lo que significa que estamos accediendo a la columna i fila j, por lo que en el loop, se debería de acceder primero a “j” y luego a “i”, además este cambio se puede realizar ya que las operaciones realizadas en el bucle más profundos de estos bucles anidados son independientes al orden de los bucles, ya que trabaja con ambos índices.

El siguiente bucle modificado ha sido el bucle que encontramos en la parte del código “Sobel edge detection”, el cual se encuentra en una situación similar a la comentada anteriormente.

Por último podemos encontrar en la parte “Denoising” la primera parte “Use salt&pepper filter” podemos encontrar dos bucles anidados los cuales utilizan índices “i” y “j”. En el bucle interno entre ambos, se anidan dos bucles más, los cuales utilizan los índices p1 y p2, ambos asignados con valores dependientes de los índices de los loops anteriores “i” y “j”. A simple vista un cambio en los índices de los bucles afectará en la escritura de datos en el array, pero como se puede comprobar después, el contenido de dicho array es ordenada por la función qsort, por lo que, finalmente concluimos que un cambio en dichos índices no modifica el resultado

**3. Obviando el Bucle 0, pruebe diferentes paralelizaciones con OpenMP comentando cuales deberían obtener mejor rendimiento.**
**a. Es imprescindible que el programa siga realizando el mismo algoritmo, por lo que solo se deberían realizar cambios en el programa que no cambien la salida.**

**b. No es necesaria la exploración completa de todas las posibles paralelizaciones, es necesario utilizar los conocimientos obtenidos en la práctica para acotar cuáles serían las mejores soluciones. Los razonamientos que utilice deben ser incluidos en la memoria.**

Hemos creado un total de 2 archivos, edgeDetector_paral1.c y edgeDetector_paral2.c en los cuales hemos probado diferentes paralelizaciones en diferentes partes del código.
En la primera implementación hemos realizado la paralelización en bucles más externos de los bucles anidados. Esta implementación debería de ser la mejor en cuanto a eficiencia.
En la segunda implementación hemos, paralelizado los bucles más auxiliares de los bucles anidados pero usando collapse(2).

**4. Rellene una tabla con resultados de tiempos y speedup respecto a la versión serie para imágenes de distintas resoluciones (SD, HD, FHD, UHD-4k, UHD-8k). Añada a su vez una columna que sea la tasa de fps a la que procesaría el programa.**

Para lograr esto, hemos creado un script, llamado time_avg.sh que ejecuta cada programa el número de veces que se desee por cada imagen y calcula la media de tiempos de las diferentes ejecuciones para cada imagen.

| Programa | SD | HD | FHD | 4k | 8k |
|---|---|---|---|---|---|
| edgeDetector | 0.032282 | 0.132217 | 0.292353 | 1.485007 | 7.508047 |
| edgeDetector-5-2 | 0.031377 | 0.125542 | 0.279842 | 1.115049 | 4.391037 |
| edgeDetector_paral1 | 0.015208 | 0.036594 | 0.056354 | 0.235354 | 0.889639 |
| edgeDetector_paral2 | 0.013008 | 0.032059 | 0.054169 | 0.209863 | 0.852746 |

La siguiente tabla representa los SpeedUp, siendo la referencia el programa original edgeDetector.
La fórmula para calcular el SpeedUp, aplicada es la siguiente:
SP = tiempo de ejecución programa original / tiempo de ejecución programa mejorado

| Programa | SD | HD | FHD | 4k | 8k |
|---|---|---|---|---|---|
| edgeDetector | 1 | 1 | 1 | 1 | 1 |
| edgeDetector-5-2 | 1.02 | 1.0531 | 1.0447 | 1.3317 | 1.7098 |
| edgeDetector_paral1 | 2.12269 | 3.6130 | 5.1877 | 6.3096 | 8.4394 |
| edgeDetector_paral2 | 2.48170 | 4.1241 | 5.39705 | 7.0760 | 8.8045 |

Como podemos comprobar se genera una muy buena mejora, programa como edgeDetector_paral1 crece de 2.12269 a 8.4394

Y por último se calcula la tasa fps a la que procesa el programa para cada imagen.
Los fps se calculan como : fps = $(\text{tiempo de ejecución})^{-1}$.

| Programa | SD | HD | FHD | 4k | 8k |
|---|---|---|---|---|---|
| edgeDetector | 30.977 | 7.56332 | 3.42052 | 0.673397 | 0.133190 |
| edgeDetector-5-2 | 31.8704 | 7.96546 | 3.57344 | 0.896821 | 0.227736 |
| edgeDetector_paral1 | 65.7548 | 27.32688 | 17.74496 | 4.248918 | 1.124051 |
| edgeDetector_paral2 | 76.8757 | 31.19248 | 18.46074 | 4.765013 | 1.172682 |

**5. Algo que hemos dejado de lado es utilizar las optimizaciones del compilador. Repita el apartado anterior añadiendo al comando gcc la bandera -O3. Investigue sobre las optimizaciones que implementa el compilador (y como podrían afectar a nuestra paralelización). ¿Implementa el desenrollado de bucles?¿Está activada esta opción? ¿Implementa algún tipo de vectorización? Nota: la opción -O3 puede generar warnings al ser compilado. Compruebe que la salida sigue siendo la misma en todo caso.**

Investigando sobre utilización de dicha flag, deducimos que por default, no realiza desenrollado de bucles, para ello habría que activar la flag -funroll-loops o funroll-all-loops

A la hora de hablar de la vectorización, llegamos a la conclusión de que si que existe algún tipo de vectorización

Tabla con tiempos de ejecucion haciendo uso de time_avg.sh:

| Programa | SD | HD | FHD | 4k | 8k |
|---|---|---|---|---|---|
| edgeDetector | .006524 | .025497 | .059469 | .462346 | 2.500703 |
| edgeDetector-5-2 | .005078 | .020903 | .046100 | .182514 | .711412 |
| edgeDetector_paral1 | .006104 | .014628 | .012259 | .047257 | .175244 |
| edgeDetector_paral2 | .007028 | .010885 | .014433 | .044637 | .186864 |

Tabla de SpeedUps

| Programa | SD | HD | FHD | 4k | 8k |
|---|---|---|---|---|---|
| edgeDetector | 1 | 1 | 1 | 1 | 1 |
| edgeDetector-5-2 | 1.2847 | 1.2197 | 1.29 | 2.5332 | 3.5151 |
| edgeDetector_paral1 | 1.0688 | 1.7430 | 4.8510 | 9.7836 | 14.2698 |
| edgeDetector_paral2 | 0.9282 | 2.3423 | 4.1203 | 10.3579 | 13.3824 |

Por último la tabla de fps

| Programa | SD | HD | FHD | 4k | 8k |
|---|---|---|---|---|---|
| edgeDetector | 153.307 | 39.130 | 16.844 | 2.164 | 0.399 |
| edgeDetector-5-2 | 196.876 | 47.822 | 21.688 | 5.477 | 1.405 |
| edgeDetector_paral1 | 163.730 | 68.392 | 81.652 | 21.130 | 5.709 |
| edgeDetector_paral2 | 142.320 | 91.941 | 69.271 | 22.395 | 5.353 |