PROGRAMACIÓN CONCURRENTE - Trabajo Práctico 2 Parte 3 (GPU)

INTEGRANTES DEL EQUIPO:

Fierro Agustin 42427695
Galindo Ricardo 33786709
Uranga Esteban 39389794
Cocciardi Agustin 40231779
Battistelli Agustin 41129795

Enunciado:

Se debe poner capturas de pantallas de las distintas ejecuciones. Los ejercicios se encuentran en el siguiente enlace

https://github.com/Progc-unlam/material-progc/blob/main/Ejemplos/GPU/TP_2_Parte_3_GPU.ipynb

Introducción:

Preguntas del TP:

a) A continuación se especifican los bloques de código donde se llevan a cabo las siguientes partes:

Reservar memoria en GPU:

```
img_O_gpu = cuda.mem_alloc( img_O_cpu.nbytes )
img_R_gpu = cuda.mem_alloc( img_R_cpu.nbytes )
```

Transferir datos de la CPU a la GPU:

```
cuda.memcpy_htod( img_O_gpu, img_O_cpu )
cuda.memcpy_htod( img_R_gpu, img_R_cpu )
```

Transferir el kernel y ejecutar el algoritmo:

```
module = SourceModule("""
#define PIXEL_ROJO( x,y) (x+(y*ancho))*3
#define PIXEL_VERDE(x,y) PIXEL_ROJO(x,y) + 1
#define PIXEL_AZUL( x,y) PIXEL_ROJO(x,y) + 2

__global__ void kernel_img( int ancho, int alto, int *img_O, int *img_R )
{
  // Calculo las coordenadas del Thread en dos dimensiones.
  int idx = threadIdx.x + blockIdx.x*blockDim.x;
  int idy = threadIdx.y + blockIdx.y*blockDim.y;
  float fGris = 0.0;

  // Verifico que los Thread, esten dentro de las dimensiones de la imagen.
  if( idx < ancho && idy < alto )
  {
    // Calculo el color gris para el pixel a partir de los componentes.
    fGris = (float)img_O[ PIXEL_ROJO(  idx, idy ) ]*0.29;  // Componente Rojo del pixel.
    fGris +=(float)img_O[ PIXEL_VERDE( idx, idy ) ]*0.59;  // Componente Verde del pixel.
    fGris +=(float)img_O[ PIXEL_AZUL(  idx, idy ) ]*0.11;  // Componente Azul del pixel.
    // Escribo el color del pixel.
    img_R[ PIXEL_ROJO(  idx, idy ) ] = (int) fGris;
    img_R[ PIXEL_VERDE( idx, idy ) ] = (int) fGris;
    img_R[ PIXEL_AZUL(  idx, idy ) ] = (int) fGris;

  }
}
""")

# ------------------------------------------------------------------------
kernel = module.get_function("kernel_img")

dim_hilo_x = 6
dim_bloque_x = int( (img_ancho+dim_hilo_x-1) / dim_hilo_x )

dim_hilo_y = 6
dim_bloque_y = int( (img_alto+dim_hilo_y-1) / dim_hilo_y )

# ------------------------------------------------------------------------
kernel( numpy.int32(img_ancho), numpy.int32(img_alto), img_O_gpu, img_R_gpu,
        block=( dim_hilo_x, dim_hilo_y, 1 ),
        grid=(dim_bloque_x, dim_bloque_y,1) )
```

Transferir datos de la GPU a la CPU:

```
cuda.memcpy_dtoh( img_R_cpu, img_R_gpu )
```

Limpiar memoria:

```
img_O_gpu.free()
img_R_gpu.free()
```

b) ¿ Cuál es la configuración inicial(por defecto) de Grilla y Bloques con que se ejecuta el algoritmo?

Los bloques utilizan dimensiones de [6,6] (definida en el propio código) mientras que la grilla utiliza dimensiones de [171, 128] (depende de las dimensiones de alto y ancho de la imagen a utilizar).

c) ¿Cuántos hilos se crean en total cuando se llama al kernel? **Tips**: utilice la formula *dim_hilo_x*dim_bloque_x*dim_hilo_y*dim_bloque_y* para calcular la cantidad de threads total

Utilizando la fórmula provista para calcular el número de hilos y sabiendo que los valores son:
dim_hilo_x = 6
dim_bloque_x = 171
dim_hilo_y = 6
dim_bloque_y = 128
Se obtiene que se utilizarán en total unos **787968** threads.

d) ¿Cuántos hilos se planifican de más? Tips: Los que no tengan condición verdadera en la condición dentro del kernel.

Dado a que cada pixel de la imagen utiliza un thread dentro de GPU, se puede calcular el número total de threads multiplicando las dimensiones de alto y ancho de la imagen. Las dimensiones en este caso son 1024 y 768, por lo que el número de threads total que serían necesarios son **786432**.
Dado a que voy a crear 787968 threads en total, si le resto los 786432 que voy a usar, obtengo un total de **1536** threads que se van a crear de más.

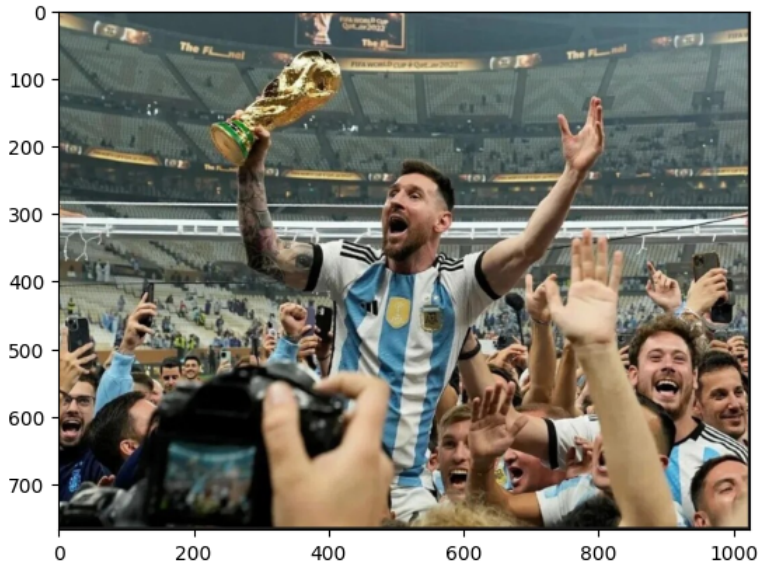
e) Modifique el código del Kernel, para que en lugar de aplicar escalas de grises a la imagen, aplique el filtro de inversión de colores.

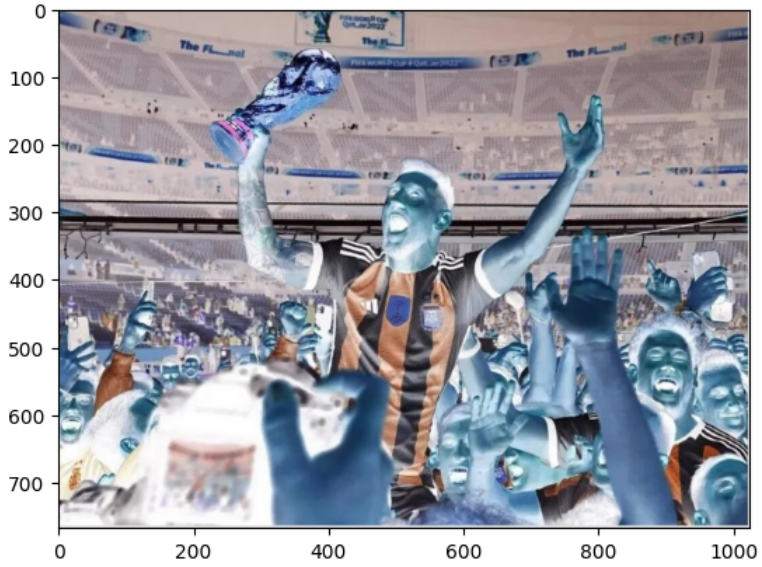
Código del algoritmo para inversión de colores:

```
__global__ void kernel_img( int ancho, int alto, int *img_O, int *img_R )
{
  // Calculo las coordenadas del Thread en dos dimensiones.
  int idx = threadIdx.x + blockIdx.x*blockDim.x;
  int idy = threadIdx.y + blockIdx.y*blockDim.y;
  int pixelRojo, pixelAzul, pixelVerde = 0;

  // Verifico que los Thread, esten dentro de las dimensiones de la imagen.
  if( idx < ancho && idy < alto )
  {
    // Calculo el color gris para el pixel a partir de los componentes.
    pixelRojo = 255-img_O[ PIXEL_ROJO(  idx, idy ) ];  // Componente Rojo del pixel.
    pixelVerde =255-img_O[ PIXEL_VERDE( idx, idy ) ];  // Componente Verde del pixel.
    pixelAzul =255-img_O[ PIXEL_AZUL(  idx, idy ) ];  // Componente Azul del pixel.
    // Escribo el color del pixel.
    img_R[ PIXEL_ROJO(  idx, idy ) ] = pixelRojo;
    img_R[ PIXEL_VERDE( idx, idy ) ] = pixelVerde;
    img_R[ PIXEL_AZUL(  idx, idy ) ] = pixelAzul;
  }
}
```

Resultado:





f) Utilice NVProf para medir la velocidad de respuesta que tiene el algoritmo durante su ejecución de acuerdo a la siguiente configuración y describa qué sucede con los tiempos.

El máximo tamaño de un bloque que soporta GPU se obtuvo con el comando previsto en el Colab.
(16,19,1):

```
==2702== NVPROF is profiling process 2702, command: python3 filter_image.py
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Grilla : [ 171 , 128 ], Bloques: [ 16 , 19  ]
==2702== Profiling application: python3 filter_image.py
==2702== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   66.14%  7.3202ms         1  7.3202ms  7.3202ms  7.3202ms  [CUDA memcpy DtoH]
                   32.11%  3.5534ms         2  1.7767ms  1.7741ms  1.7792ms  [CUDA memcpy HtoD]
                    1.75%  193.47us         1  193.47us  193.47us  193.47us  kernel_img
      API calls:   83.27%  228.26ms         1  228.26ms  228.26ms  228.26ms  cuCtxCreate
                   11.43%  31.337ms         1  31.337ms  31.337ms  31.337ms  cuCtxDetach
                    3.44%  9.4221ms         1  9.4221ms  9.4221ms  9.4221ms  cuMemcpyDtoH
                    1.47%  4.0283ms         2  2.0141ms  1.9302ms  2.0981ms  cuMemcpyHtoD
                    0.22%  601.66us         2  300.83us  134.40us  467.26us  cuMemFree
                    0.09%  259.73us         2  129.87us  92.007us  167.73us  cuMemAlloc
                    0.03%  87.116us         1  87.116us  87.116us  87.116us  cuModuleLoadDataEx
                    0.02%  44.795us         1  44.795us  44.795us  44.795us  cuModuleUnload
                    0.01%  39.724us         1  39.724us  39.724us  39.724us  cuLaunchKernel
                    0.00%  7.3020us         1  7.3020us  7.3020us  7.3020us  cuDeviceGetPCIBusId
                    0.00%  4.2420us         2  2.1210us     605ns  3.6370us  cuCtxPopCurrent
                    0.00%  3.3920us         2  1.6960us     244ns  3.1480us  cuCtxPushCurrent
                    0.00%  2.8150us         3     938ns     677ns  1.2300us  cuDeviceGetAttribute
                    0.00%  2.7050us         3     901ns     299ns  1.5250us  cuDeviceGetCount
                    0.00%  1.8160us         1  1.8160us  1.8160us  1.8160us  cuModuleGetFunction
                    0.00%  1.7850us         2     892ns     746ns  1.0390us  cuCtxGetDevice
                    0.00%  1.2190us         2     609ns     488ns     731ns  cuDeviceGet
                    0.00%  1.1110us         1  1.1110us  1.1110us  1.1110us  cuDeviceComputeCapability
                    0.00%     731ns         1     731ns     731ns     731ns  cuFuncSetBlockShape
```

(24,24,1):

```
==2986== NVPROF is profiling process 2986, command: python3 filter_image.py
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Grilla : [ 171 , 128 ], Bloques: [ 24 , 24  ]
==2986== Profiling application: python3 filter_image.py
==2986== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   60.89%  6.5637ms         1  6.5637ms  6.5637ms  6.5637ms  [CUDA memcpy DtoH]
                   34.26%  3.6927ms         2  1.8464ms  1.8398ms  1.8530ms  [CUDA memcpy HtoD]
                    4.85%  522.91us         1  522.91us  522.91us  522.91us  kernel_img
      API calls:   83.52%  216.68ms         1  216.68ms  216.68ms  216.68ms  cuCtxCreate
                   10.95%  28.412ms         1  28.412ms  28.412ms  28.412ms  cuCtxDetach
                    3.44%  8.9202ms         1  8.9202ms  8.9202ms  8.9202ms  cuMemcpyDtoH
                    1.63%  4.2185ms         2  2.1093ms  2.0489ms  2.1696ms  cuMemcpyHtoD
                    0.24%  618.44us         2  309.22us  140.78us  477.66us  cuMemFree
                    0.15%  377.80us         2  188.90us  158.20us  219.60us  cuMemAlloc
                    0.05%  121.11us         1  121.11us  121.11us  121.11us  cuModuleLoadDataEx
                    0.02%  43.223us         1  43.223us  43.223us  43.223us  cuModuleUnload
                    0.01%  25.725us         1  25.725us  25.725us  25.725us  cuLaunchKernel
                    0.00%  7.6660us         2  3.8330us     899ns  6.7670us  cuCtxGetDevice
                    0.00%  6.2930us         1  6.2930us  6.2930us  6.2930us  cuDeviceGetPCIBusId
                    0.00%  4.1230us         2  2.0610us     377ns  3.7460us  cuCtxPopCurrent
                    0.00%  3.6290us         2  1.8140us     180ns  3.4490us  cuCtxPushCurrent
                    0.00%  3.0520us         3  1.0170us     267ns  1.5360us  cuDeviceGetCount
                    0.00%  2.7470us         3     915ns     723ns  1.1440us  cuDeviceGetAttribute
                    0.00%  1.3140us         2     657ns     436ns     878ns  cuDeviceGet
                    0.00%  1.2760us         1  1.2760us  1.2760us  1.2760us  cuModuleGetFunction
                    0.00%     884ns         1     884ns     884ns     884ns  cuDeviceComputeCapability
                    0.00%     867ns         1     867ns     867ns     867ns  cuFuncSetBlockShape
```

Máximo soportado por GPU (1024, 1024, 64):

```
==3396== NVPROF is profiling process 3396, command: python3 filter_image.py
Traceback (most recent call last):
  File "/content/filter_image.py", line 69, in <module>
    kernel( numpy.int32(img_ancho), numpy.int32(img_alto), img_O_gpu, img_R_gpu,
  File "/usr/local/lib/python3.10/dist-packages/pycuda/driver.py", line 481, in function_call
    func._set_block_shape(*block)
pycuda._driver.LogicError: cuFuncSetBlockShape failed: invalid argument
==3396== Profiling application: python3 filter_image.py
==3396== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  3.5973ms         2  1.7986ms  1.7906ms  1.8067ms  [CUDA memcpy HtoD]
      API calls:   86.46%  223.55ms         1  223.55ms  223.55ms  223.55ms  cuCtxCreate
                   11.52%  29.794ms         1  29.794ms  29.794ms  29.794ms  cuCtxDetach
                    1.57%  4.0683ms         2  2.0341ms  1.9558ms  2.1125ms  cuMemcpyHtoD
                    0.26%  663.15us         2  331.58us  199.04us  464.11us  cuMemFree
                    0.12%  312.24us         2  156.12us  99.734us  212.51us  cuMemAlloc
                    0.04%  114.05us         1  114.05us  114.05us  114.05us  cuModuleLoadDataEx
                    0.01%  21.968us         1  21.968us  21.968us  21.968us  cuModuleUnload
                    0.00%  7.7150us         2  3.8570us     856ns  6.8590us  cuCtxGetDevice
                    0.00%  6.7070us         1  6.7070us  6.7070us  6.7070us  cuDeviceGetPCIBusId
                    0.00%  3.4380us         3  1.1460us     276ns  1.6500us  cuDeviceGetCount
                    0.00%  3.1510us         4     787ns     210ns  2.2940us  cuCtxPopCurrent
                    0.00%  2.8130us         4     703ns     158ns  1.9320us  cuCtxPushCurrent
                    0.00%  2.7740us         3     924ns     638ns  1.2760us  cuDeviceGetAttribute
                    0.00%  1.8670us         1  1.8670us  1.8670us  1.8670us  cuModuleGetFunction
                    0.00%  1.3270us         2     663ns     624ns     703ns  cuDeviceGet
                    0.00%  1.0230us         1  1.0230us  1.0230us  1.0230us  cuDeviceComputeCapability
                    0.00%     872ns         1     872ns     872ns     872ns  cuFuncSetBlockShape
                    0.00%     770ns         1     770ns     770ns     770ns  cuGetErrorString
======== Error: Application returned non-zero code 1
```

Al usar una dimensión de bloque mucho mayor que la que se había definido inicialmente, el tiempo de ejecución del kernel no se ve reducido, esto porque ya se había visto que al utilizar una dimensión de bloque de (6,6) el número de hilos creados terminó siendo superior al número que iban a utilizarse. Incrementar la dimensión de bloque no hará que los hilos creados trabajen más rápido, solamente tendré más hilos ociosos en tiempo de ejecución.

g) En el punto anterior ¿Qué sucedió al medir el tiempo con máximo de Threads soportados por la GPU?

Se produjo un error debido a que la forma de bloque no es válida para la función de kernel, siendo que se está trabajando en dos dimensiones (X e Y), y para obtener el máximo de threads que soporta la GPU es necesario disponer de la dimensión Z. Pero revisando lo ocurrido al aumentar el número de hilos en X e Y, y viendo que eso no produce aceleraciones en el tiempo, asumimos que utilizando la máxima cantidad de hilos, su tiempo sería superior al tiempo de ejecución con dimensiones (24,24,1).

h) Mida el tiempo de respuesta que tiene el algoritmo con la dimension "X" en su maxima capacidad y el resto en 1 (ej:(1024,1,1)). Luego modifique la configuración para que la dimension "Y" tenga su maxima capacidad y el resto en 1.(ej:(1,1024,1)) Compare los resultado. ¿A que cree que se debe la diferencia de tiempo?

(1024,1,1):

```
==6418== NVPROF is profiling process 6418, command: python3 filter_image.py
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Grilla : [ 171 , 128 ], Bloques: [ 1024 , 1  ]
==6418== Profiling application: python3 filter_image.py
==6418== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   66.31%  9.3126ms         1  9.3126ms  9.3126ms  9.3126ms  [CUDA memcpy DtoH]
                   29.74%  4.1765ms         2  2.0882ms  1.9308ms  2.2456ms  [CUDA memcpy HtoD]
                    3.95%  555.04us         1  555.04us  555.04us  555.04us  kernel_img
      API calls:   83.18%  278.10ms         1  278.10ms  278.10ms  278.10ms  cuCtxCreate
                   11.33%  37.880ms         1  37.880ms  37.880ms  37.880ms  cuCtxDetach
                    3.65%  12.209ms         1  12.209ms  12.209ms  12.209ms  cuMemcpyDtoH
                    1.42%  4.7587ms         2  2.3794ms  2.1382ms  2.6206ms  cuMemcpyHtoD
                    0.21%  712.00us         2  356.00us  205.38us  506.62us  cuMemFree
                    0.13%  431.79us         2  215.89us  155.40us  276.39us  cuMemAlloc
                    0.04%  136.67us         1  136.67us  136.67us  136.67us  cuModuleLoadDataEx
                    0.02%  52.782us         1  52.782us  52.782us  52.782us  cuModuleUnload
                    0.01%  35.048us         1  35.048us  35.048us  35.048us  cuLaunchKernel
                    0.00%  9.0220us         1  9.0220us  9.0220us  9.0220us  cuDeviceGetPCIBusId
                    0.00%  5.7550us         1  5.7550us  5.7550us  5.7550us  cuModuleGetFunction
                    0.00%  4.4960us         2  2.2480us     772ns  3.7240us  cuCtxPopCurrent
                    0.00%  4.2280us         2  2.1140us     354ns  3.8740us  cuCtxPushCurrent
                    0.00%  3.3750us         3  1.1250us     362ns  1.6580us  cuDeviceGetCount
                    0.00%  2.1070us         2  1.0530us     938ns  1.1690us  cuCtxGetDevice
                    0.00%  1.6470us         3     549ns     275ns  1.0380us  cuDeviceGetAttribute
                    0.00%  1.3100us         1  1.3100us  1.3100us  1.3100us  cuDeviceComputeCapability
                    0.00%  1.2870us         2     643ns     643ns     644ns  cuDeviceGet
                    0.00%     981ns         1     981ns     981ns     981ns  cuFuncSetBlockShape
```

(1,1024,1):

```
==6706== NVPROF is profiling process 6706, command: python3 filter_image.py
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Grilla : [ 171 , 128 ], Bloques: [ 1 , 1024  ]
==6706== Profiling application: python3 filter_image.py
==6706== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   63.60%  7.6545ms         1  7.6545ms  7.6545ms  7.6545ms  [CUDA memcpy DtoH]
                   31.03%  3.7345ms         2  1.8673ms  1.7965ms  1.9380ms  [CUDA memcpy HtoD]
                    5.37%  645.82us         1  645.82us  645.82us  645.82us  kernel_img
      API calls:   83.83%  233.05ms         1  233.05ms  233.05ms  233.05ms  cuCtxCreate
                   10.55%  29.339ms         1  29.339ms  29.339ms  29.339ms  cuCtxDetach
                    3.69%  10.248ms         1  10.248ms  10.248ms  10.248ms  cuMemcpyDtoH
                    1.49%  4.1548ms         2  2.0774ms  1.9469ms  2.2079ms  cuMemcpyHtoD
                    0.23%  642.68us         2  321.34us  177.40us  465.28us  cuMemFree
                    0.11%  308.48us         2  154.24us  101.46us  207.02us  cuMemAlloc
                    0.05%  142.11us         1  142.11us  142.11us  142.11us  cuModuleLoadDataEx
                    0.02%  45.796us         1  45.796us  45.796us  45.796us  cuModuleUnload
                    0.01%  31.752us         1  31.752us  31.752us  31.752us  cuLaunchKernel
                    0.00%  6.9160us         1  6.9160us  6.9160us  6.9160us  cuModuleGetFunction
                    0.00%  6.8420us         1  6.8420us  6.8420us  6.8420us  cuDeviceGetPCIBusId
                    0.00%  4.7510us         2  2.3750us     497ns  4.2540us  cuCtxPopCurrent
                    0.00%  3.9030us         3  1.3010us     376ns  2.0950us  cuDeviceGetCount
                    0.00%  3.5730us         2  1.7860us     231ns  3.3420us  cuCtxPushCurrent
                    0.00%  2.8750us         3     958ns     691ns  1.1790us  cuDeviceGetAttribute
                    0.00%  1.6950us         2     847ns     806ns     889ns  cuCtxGetDevice
                    0.00%  1.2210us         2     610ns     529ns     692ns  cuDeviceGet
                    0.00%  1.0040us         1  1.0040us  1.0040us  1.0040us  cuFuncSetBlockShape
                    0.00%     951ns         1     951ns     951ns     951ns  cuDeviceComputeCapability
```

La diferencia de tiempo se debe a la agrupación de los threads en unidades de ejecución. Si se elige una dimensión de (1024,1,1) el máximo número de threads se agrupan dentro de una sola unidad. En cambio, si se elige la dimensión de (1,1024,1) se tendrán un total de 1024 unidades de ejecución utilizando únicamente un hilo cada una. Lo que hará al algoritmo menos eficiente que si se utilizaran los máximos hilos en la dimensión de X.

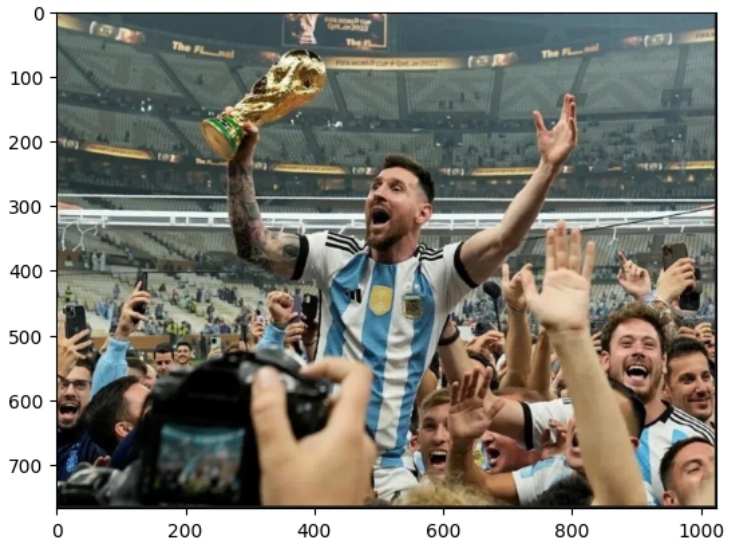
i) ¿Qué cambios realizaría para que solo se procese la parte derecha de la imagen? ¿Qué sucede con la parte izquierda en la imagen resultante? ¿Qué sucede con la velocidad de respuesta?

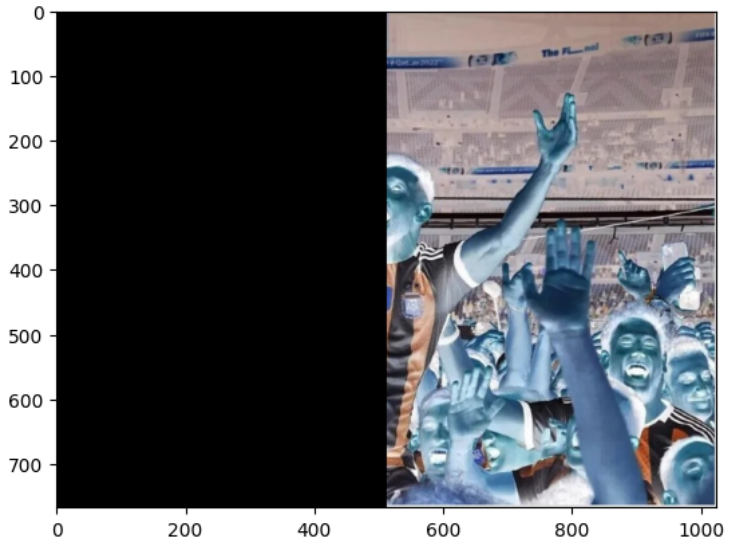
Los cambios que realizaría para procesar solo la parte derecha de la imagen serían los siguientes:

```
if( idx < ancho && idy < alto )
{
  // Calculo el color gris para el pixel a partir de los componentes.
  if (idx > (ancho/2))
  {
    pixelRojo = 255-img_O[ PIXEL_ROJO(  idx, idy ) ];  // Componente Rojo del pixel.
    pixelVerde =255-img_O[ PIXEL_VERDE( idx, idy ) ];  // Componente Verde del pixel.
    pixelAzul =255-img_O[ PIXEL_AZUL(  idx, idy ) ];  // Componente Azul del pixel.
    // Escribo el color del pixel.
    img_R[ PIXEL_ROJO(  idx, idy ) ] = pixelRojo;
    img_R[ PIXEL_VERDE( idx, idy ) ] = pixelVerde;
    img_R[ PIXEL_AZUL(  idx, idy ) ] = pixelAzul;
  }
  else
  {
    pixelRojo = img_O[ PIXEL_ROJO(  idx, idy ) ];  // Componente Rojo del pixel.
    pixelVerde =img_O[ PIXEL_VERDE( idx, idy ) ];  // Componente Verde del pixel.
    pixelAzul =img_O[ PIXEL_AZUL(  idx, idy ) ];  // Componente Azul del pixel.
    // Escribo el color del pixel.
    img_R[ PIXEL_ROJO(  idx, idy ) ] = pixelRojo;
    img_R[ PIXEL_VERDE( idx, idy ) ] = pixelVerde;
    img_R[ PIXEL_AZUL(  idx, idy ) ] = pixelAzul;
  }
}
```

Para procesar los píxeles de la parte derecha, tengo que preguntar si el índice de x es mayor a la mitad del ancho de la imagen. Aunque también tengo que colocar un else para setear los píxeles de la mitad izquierda de la imagen iguales a su color original, de lo contrario, la parte izquierda me quedará de negro al no tener como asignarle el color a los píxeles.

Resultado cuando aplico el filtro a la parte derecha y no hago nada con la parte izquierda:





Resultado cuando aplico el filtro a la parte derecha y proceso la parte izquierda para dejarla igual a la original:

Tiempo de respuesta cuando aplico el filtro a toda la imagen:

```
==9130== NVPROF is profiling process 9130, command: python3 filter_image.py
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Grilla : [ 171 , 128 ], Bloques: [ 6 , 6  ]
==9130== Profiling application: python3 filter_image.py
==9130== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   61.97%  6.7206ms         1  6.7206ms  6.7206ms  6.7206ms  [CUDA memcpy DtoH]
                   36.76%  3.9864ms         2  1.9932ms  1.9492ms  2.0373ms  [CUDA memcpy HtoD]
                    1.27%  137.41us         1  137.41us  137.41us  137.41us  kernel_img
      API calls:   83.59%  219.60ms         1  219.60ms  219.60ms  219.60ms  cuCtxCreate
                   10.95%  28.767ms         1  28.767ms  28.767ms  28.767ms  cuCtxDetach
                    3.31%  8.6842ms         1  8.6842ms  8.6842ms  8.6842ms  cuMemcpyDtoH
                    1.72%  4.5088ms         2  2.2544ms  2.2300ms  2.2788ms  cuMemcpyHtoD
                    0.23%  614.48us         2  307.24us  142.36us  472.12us  cuMemFree
                    0.13%  345.27us         2  172.63us  155.27us  190.00us  cuMemAlloc
                    0.04%  93.324us         1  93.324us  93.324us  93.324us  cuModuleLoadDataEx
                    0.02%  43.036us         1  43.036us  43.036us  43.036us  cuModuleUnload
                    0.01%  28.552us         1  28.552us  28.552us  28.552us  cuLaunchKernel
                    0.00%  6.2040us         1  6.2040us  6.2040us  6.2040us  cuDeviceGetPCIBusId
                    0.00%  4.6890us         1  4.6890us  4.6890us  4.6890us  cuModuleGetFunction
                    0.00%  3.6760us         2  1.8380us     420ns  3.2560us  cuCtxPopCurrent
                    0.00%  3.3510us         2  1.6750us     196ns  3.1550us  cuCtxPushCurrent
                    0.00%  2.8540us         2  1.4270us     559ns  2.2950us  cuDeviceGet
                    0.00%  2.6890us         3     896ns     668ns  1.2210us  cuDeviceGetAttribute
                    0.00%  2.2280us         2  1.1140us  1.0110us  1.2170us  cuCtxGetDevice
                    0.00%  2.2240us         3     741ns     292ns  1.0070us  cuDeviceGetCount
                    0.00%     904ns         1     904ns     904ns     904ns  cuFuncSetBlockShape
                    0.00%     875ns         1     875ns     875ns     875ns  cuDeviceComputeCapability
```

Tiempo de respuesta cuando aplico el filtro solo a la parte derecha (y me aseguro que la parte izquierda no quede de color negro):

```
==11788== NVPROF is profiling process 11788, command: python3 filter_image.py
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Grilla : [ 171 , 128 ], Bloques: [ 6 , 6  ]
==11788== Profiling application: python3 filter_image.py
==11788== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   66.27%  7.1412ms         1  7.1412ms  7.1412ms  7.1412ms  [CUDA memcpy DtoH]
                   32.46%  3.4981ms         2  1.7491ms  1.7396ms  1.7585ms  [CUDA memcpy HtoD]
                    1.27%  137.18us         1  137.18us  137.18us  137.18us  kernel_img
      API calls:   82.11%  202.36ms         1  202.36ms  202.36ms  202.36ms  cuCtxCreate
                   12.10%  29.812ms         1  29.812ms  29.812ms  29.812ms  cuCtxDetach
                    3.70%  9.1230ms         1  9.1230ms  9.1230ms  9.1230ms  cuMemcpyDtoH
                    1.62%  3.9951ms         2  1.9975ms  1.9383ms  2.0567ms  cuMemcpyHtoD
                    0.26%  637.65us         2  318.83us  149.60us  488.06us  cuMemFree
                    0.11%  282.20us         2  141.10us  95.191us  187.01us  cuMemAlloc
                    0.05%  122.08us         1  122.08us  122.08us  122.08us  cuModuleLoadDataEx
                    0.02%  56.883us         1  56.883us  56.883us  56.883us  cuModuleUnload
                    0.01%  33.808us         1  33.808us  33.808us  33.808us  cuLaunchKernel
                    0.00%  7.8020us         2  3.9010us     699ns  7.1030us  cuCtxGetDevice
                    0.00%  7.0580us         1  7.0580us  7.0580us  7.0580us  cuDeviceGetPCIBusId
                    0.00%  5.4130us         2  2.7060us     447ns  4.9660us  cuCtxPopCurrent
                    0.00%  4.7430us         2  2.3710us     199ns  4.5440us  cuCtxPushCurrent
                    0.00%  2.5950us         3     865ns     681ns  1.2070us  cuDeviceGetAttribute
                    0.00%  2.4160us         3     805ns     231ns  1.1160us  cuDeviceGetCount
                    0.00%  1.9340us         1  1.9340us  1.9340us  1.9340us  cuModuleGetFunction
                    0.00%  1.3650us         2     682ns     545ns     820ns  cuDeviceGet
                    0.00%  1.1130us         1  1.1130us  1.1130us  1.1130us  cuFuncSetBlockShape
                    0.00%     939ns         1     939ns     939ns     939ns  cuDeviceComputeCapability
```

Tiempo de respuesta cuando aplico el filtro solo a la parte derecha (y dejo la parte izquierda sin procesar):

```
==12030== NVPROF is profiling process 12030, command: python3 filter_image.py
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Imagen del filtro:  imagen.jpg  - tipo  RGB - [ 1024 ,  768 ]
Grilla : [ 171 , 128 ], Bloques: [ 6 , 6  ]
==12030== Profiling application: python3 filter_image.py
==12030== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:   64.38%  7.0414ms         1  7.0414ms  7.0414ms  7.0414ms  [CUDA memcpy DtoH]
                   34.95%  3.8226ms         2  1.9113ms  1.8109ms  2.0118ms  [CUDA memcpy HtoD]
                    0.68%  74.016us         1  74.016us  74.016us  74.016us  kernel_img
      API calls:   81.91%  203.20ms         1  203.20ms  203.20ms  203.20ms  cuCtxCreate
                   12.27%  30.439ms         1  30.439ms  30.439ms  30.439ms  cuCtxDetach
                    3.57%  8.8602ms         1  8.8602ms  8.8602ms  8.8602ms  cuMemcpyDtoH
                    1.73%  4.2980ms         2  2.1490ms  1.9661ms  2.3319ms  cuMemcpyHtoD
                    0.27%  681.05us         2  340.52us  195.91us  485.14us  cuMemFree
                    0.13%  322.91us         2  161.46us  99.621us  223.29us  cuMemAlloc
                    0.06%  159.93us         1  159.93us  159.93us  159.93us  cuModuleLoadDataEx
                    0.02%  41.566us         1  41.566us  41.566us  41.566us  cuModuleUnload
                    0.01%  30.953us         1  30.953us  30.953us  30.953us  cuLaunchKernel
                    0.00%  7.0670us         1  7.0670us  7.0670us  7.0670us  cuDeviceGetPCIBusId
                    0.00%  5.3800us         1  5.3800us  5.3800us  5.3800us  cuModuleGetFunction
                    0.00%  4.5320us         2  2.2660us     518ns  4.0140us  cuCtxPopCurrent
                    0.00%  3.3240us         2  1.6620us     192ns  3.1320us  cuCtxPushCurrent
                    0.00%  2.5750us         3     858ns     269ns  1.1610us  cuDeviceGetCount
                    0.00%  1.9690us         3     656ns     484ns     988ns  cuDeviceGetAttribute
                    0.00%  1.5360us         2     768ns     743ns     793ns  cuCtxGetDevice
                    0.00%  1.2730us         2     636ns     497ns     776ns  cuDeviceGet
                    0.00%     938ns         1     938ns     938ns     938ns  cuDeviceComputeCapability
                    0.00%     694ns         1     694ns     694ns     694ns  cuFuncSetBlockShape
```

Cada uno de los threads trabaja con un píxel de la imagen. Si quiero trabajar solo con una mitad del ancho de la imagen, la mitad de los threads no van a realizar ninguna tarea, por lo cual el tiempo se reduce casi por la mitad al usar el algoritmo.

Link a repositorio de Github:

Conclusiones:

Una dificultad que afrontamos fue el seteo del entorno de ejecución. Ya que no teníamos seteado el acelerador por hardware como GPU, y al momento de ejecutar el desarrollo no se reconocía el módulo driver de la biblioteca pycuda y nos arrojaba error.

Además, otra de las dificultades que surgió en la realización de este tp fue implementar el algoritmo que se encarga de la inversión de los colores de la imagen. No por la complejidad del mismo, sino porque pensábamos que consistía en ir guardando los valores de los píxeles rojo, verde y azul en una variable de forma acumulativa, cuando lo que debíamos hacer era usar un valor entero para la conversión de los colores de cada pixel para la imagen.

Bibliografía: