Práctica 1

Box Filter de una imagen

Procesadores Gráficos y Aplicaciones en Tiempo Real

Alejandro López Vizuete
Exp - 410

1. Descripción de código

__global__ void box_filter

Kernel que realiza la fusión del canal de color dado con el filtro escogido. Para ello, he escogido la forma de acceder a los canales como la función dada en el ejercicio (recombineChannels). Creo una variable mitad, que es la mitad del tamaño del filtro, para poder utilizarlo al recorrer las matrices.

Para poder fusionar el canal con el filtro, creo dos bucles for anidados desde cero hasta el tamaño del filtro. Como el thread está centrado en el pixel del centro, tengo que sumar i (o j, dependiendo de si hablamos de posición X o Y) y restar la variable mitad. Así consigo que, aunque el thread apunte al pixel del centro, me pueda mover en la cuadricula del tamaño del filtro.

Con esto, accedo a la posición del canal correspondiente, que multiplicado por la posición (i,j) del filtro, consigo que en la variable Pixel se guarde el valor correspondiente para el canal introducido.

__global__ void separateChannels

Kernel que se encarga de recibir una imagen pasada como parámetro, y separar cada pixel en los 3 canales RGB.

Este apartado es necesario realizarlo entero. Para ello, he escogido acceder a los canales e imagen como el apartado anterior. Por lo tanto, lo único necesario de realizar es meter el valor RBG (XYZ) de cada pixel, en la misma posición en su canal correspondiente.

__global__ void recombineChannels

Kernel que combina el color de los canales RGB a la imagen que se creará. En este apartado, no es necesario realizar ningún ejercicio.

void allocateMemoryAndCopyToGPU

Función que se encarga de reservar memoria para los canales de colores y el filtro. Como ejercicio, he reservado memoria de tamaño matriz float para el filtro, y además he copiado los datos de h_filter a d_filter para tener la matriz de filtro en GPU.

void create_filter

Funcion encargada de elegir el filtro que se aplicará a la imagen. Venían de serie el filtro Gaussiano 5x5 y Laplace 5x5. Adicionalmente, tal y como se pedía en el enunciado, he añadido, comentados, varios filtros más que funcionan correctamente. Esos son Nitidez 3x3, Detección de bordes 3x3, Suavizado 3x3.

void convolution

La función principal del programa. Se encarga de calcular los tamaños del Grid y del Block, y realizar las respectivas ejecuciones de los kernel declarados anteriormente. Como ejercicio, he calculado los tamaños de bloque a 32, después de diversas pruebas, y el tamaño del Grid en función del tamaño de la imagen tal como se pedía en el enunciado.

Posteriormente, he realizado las ejecuciones de los kernel con los tamaños indicados, y pasándoles los parámetros adecuados. Entre cada ejecución de kernel, realizo un cudaDeviceSynchronize y un cudaGetLastError para el correcto funcionamiento de la práctica.

void cleanup

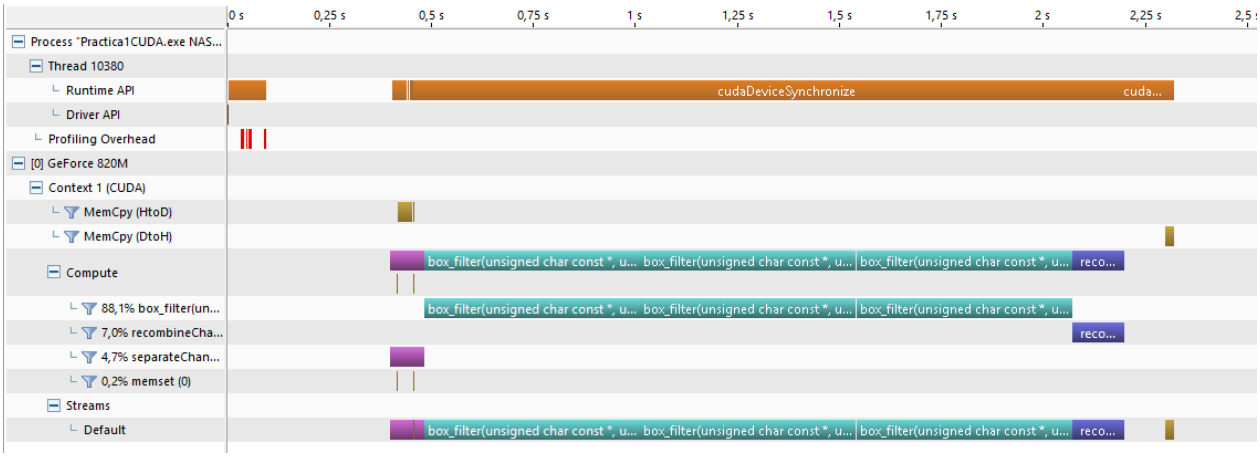
Función que libera toda la memoria reservada en la aplicación. Como ejercicio en este apartado, se pedía liberar memoria de los arrays que he declarado en memoria en la función ya descrita. Por lo que he realizado un cudaFree de la variable d_filter.

2. *Análisis de mejoras introducidas*

Mejoras de Rendimiento

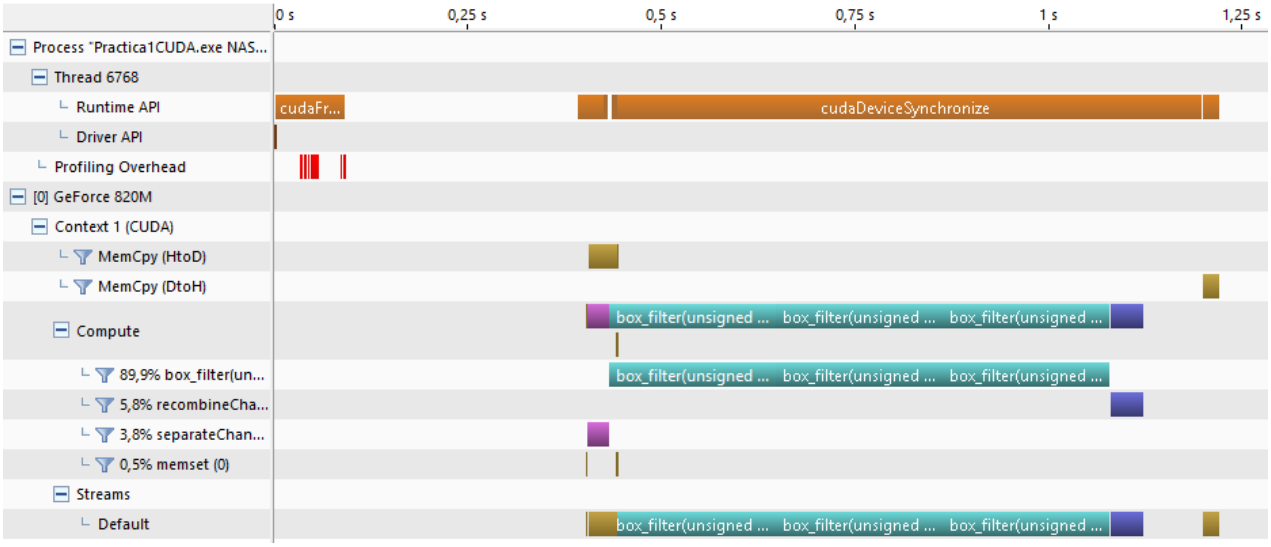
Antes de empezar, quiero comentar que al ser mi ordenador algo antiguo, al ejecutar me daba el Profiler, me daba el problema de “Warning: Unified Memory Profiling is not supported on devices of compute capability less than 3.0”.

En principio tenia puesto el tamaño de los bloques del mismo tamaño que el filtro, es decir, bloques de tamaño 5. Esto ocasionaba una gran ineficiencia como se puede ver en la siguiente imagen. Tardaba casi 2.25 segundos en tardar de ejecutarse (1800 msecs aprox.).

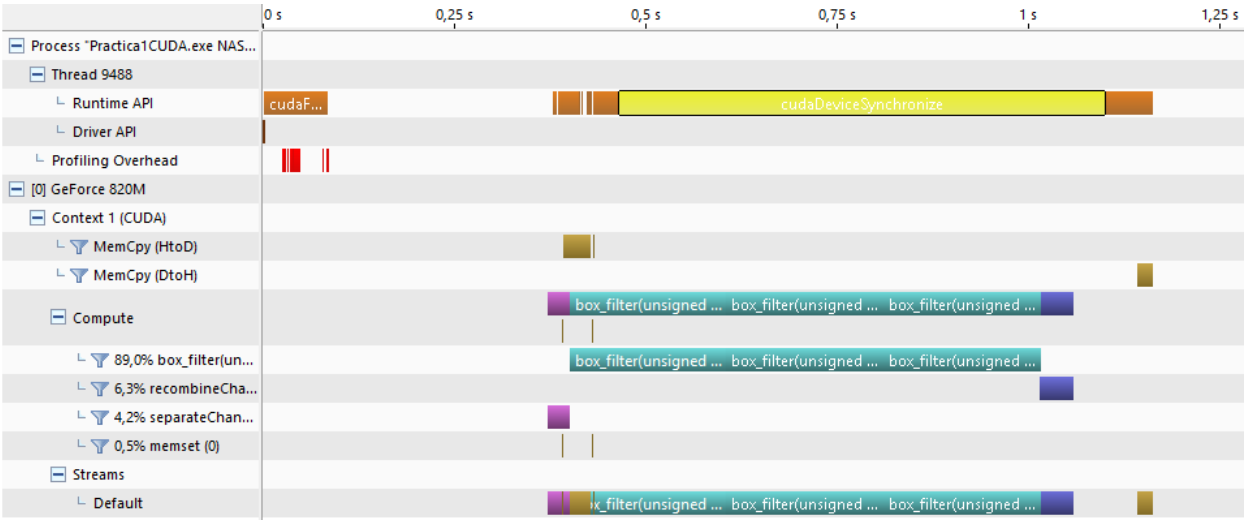


Tras revisar el temario y repasar algunos ejercicios realizados en clase, cambié el tamaño del bloque, pero no tenia clase si decantarme por un tamaño de 16, o de 32. Por lo que finalmente realicé pruebas y estos fueron los resultados.

Para tamaño de bloque 16.



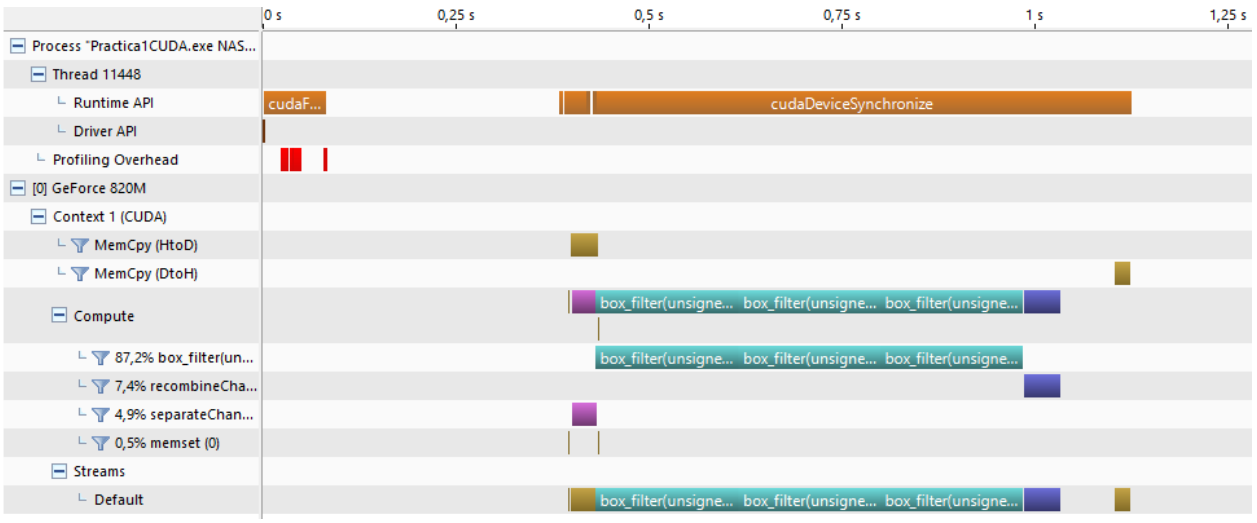
Para tamaño de bloque 32.



Como se puede observar, el tiempo de ejecución se redujo algo mas de un segundo, siendo una duración total de casi 1.25 segundos (750 msces aprox.), siendo algo más eficiente con un tamaño de 32 bloques.

Por último, como comentaré en los problemas encontrados, para poder utilizar el filtro en los bordes de la imagen, utilicé unos IFs dentro del Kernel, reduciendo así su eficiencia. Pero hablando con compañeros me dijeron que se podía utilizar las funciones min y max de CUDA para mejorar el tiempo.

Dicho esto, con una comparativa entre el programa con el mismo numero de bloques (32, ver imagen anterior), pero diferente forma de solucionar el problema de bordes, se puede observar una ligera mejora en el rendimiento del problema.



3. Comentarios personales

Problemas encontrados

El primer problema importante lo he tenido nada mas empezar con la práctica, y es la instalación de OpenCV. Lo intenté de diversas formas y no era capaz de poder hacer que me compilase. Un compañero me comentó que lo había configurado de una forma algo diferente a como viene explicado en aula virtual, por lo que usando su configuración pude crearme un proyecto ya compilado.

El segundo problema es mas bien un despiste por mi cuenta, pero como perdí dos días enteros intentado ver la solución, he creído conveniente comentarlo aquí.

El problema era que siempre me salía la imagen en negro. Cambie mucho código para ver si era algo que puse mal, hice de varias bastantes formas el kernel separateChannel, incluso pasé en box_filter, el mismo valor del canal entrarte al saliente, para que me saliera la misma imagen. Nada, siempre me salía la imagen en negro. El fallo fue que al realizar la llamada al kernel de separateChannel, estaba enviando como parámetro outputImage y no inputImage.

Otro problema que tuve, y tardé un poco en darme cuenta, fue a la hora de crear el tamaño del Grid, debido a que salía siempre alguna parte de la imagen en negro. El fallo era que tenia puestos numCols y numRows al revés.

Uno de los problemas más grandes lo he tenido con los bordes de la imagen. En un principio no me había fijado que me salían mal, pero si hacia zoom me daba cuenta de que salían de un color equivocado.

He logrado solucionarlo mediante IFs dentro del bucle de FORs, pero me di cuenta que aumentaba en casi el doble el tiempo que tardaba la aplicación en terminar. De unos 400/500 msecs a casi 750/800 msecs (en modo debug). Con esta solución tengo cuatro ifs, una por cada lado de la imagen, y si compruebo que el pixel elegido está fuera de la imagen, fuerzo a que sea el pixel del borde.

Otra idea que tuve fue crear unas variables initX e initY, inicializadas a cero, y que fueran siempre el empiece en los bucles FOR. De tal forma, cuando detectaba que se estaba el thread estaba en un pixel con una distancia inferior a la mitad del tamaño del filtro, cambiaba los valores de initX e initY para “recortar” la matriz con la que se multiplicaba el canal y el filtro. Pero tampoco me dio buen resultado.

Finalmente, hablando con algunos compañeros y dándole otro planteamiento a la idea de meter IFs dentro de los FOR, he conseguido una solución mas eficiente con la utilización de las funciones min y max, recortando el tiempo de ejecución a unos 600/650 msecs (en modo debug).

El último problema grande que he tenido es a la hora de realizar el ejercicio con memoria compartida. He llegado a crear dos arrays de memoria compartida, uno de ellos con los datos del filtro y otro con la posición del pixel actual, pero por más que lo rediseñaba no era capaz de que me saliera algún color en la imagen.

Propuestas

Una propuesta para próximos cursos, es cambiar la guía de instalación de OpenCV, debido a que en Configuration Propierties -> VC++ Directories, se puede configurar el proyecto con OpenCV sin tener que tocar el Path de Windows.

A raíz de estos, sería posible dar el proyecto ya configurado para openCV, teniendo que cambiar únicamente la raíz de los elementos de configuración del proyecto.