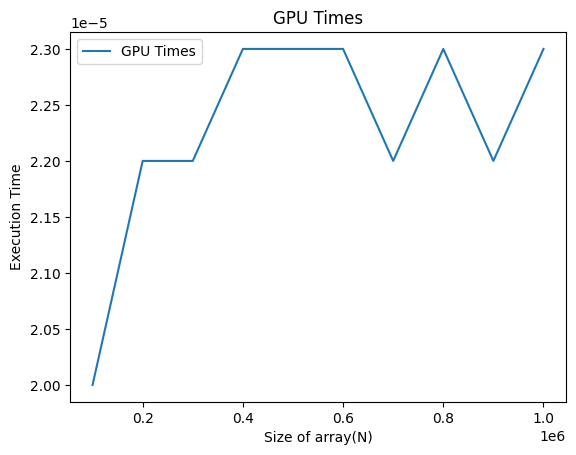
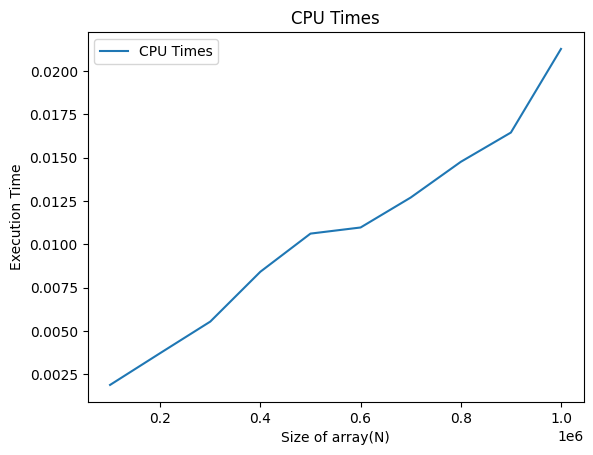
**ASIGNATURA Computación de altas prestaciones**

**Task 2**

**GPU Programming**

**Carlos Riveira Ramos y Pablo Sanchez Fernandez del Pozo**

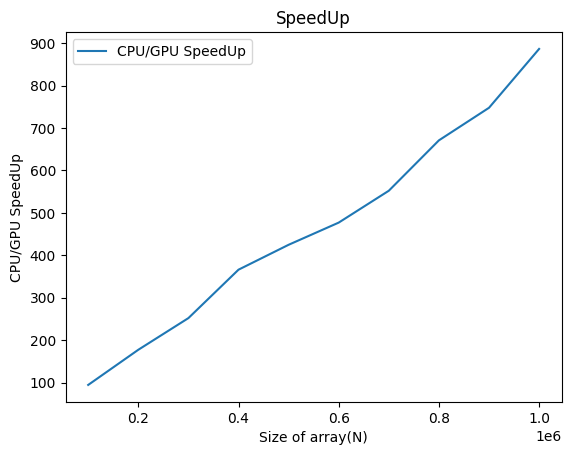
You must write a report answering the questions proposed in each exercise, plus the requested files. Submit a zip file through Moodle. Check submission date in Moodle (deadline is until 11:59 pm of that date).

* Exercise 1:
  + Compare the execution time for different values of N (array size): from 100.000 to 1.000.000 in steps of 100.000. Plot the result in a graph. Explain the results.

Podemos observar que los tiempos de ejecución del programa lanzado en la GPU es mucho más óptimo, en órdenes de magnitud (aproximadamente hasta 1000 veces más eficaz) comparado con el lanzado en CPU.

También es remarcable el incremento de los tiempos en CPU frente a GPU, en este primero los tiempos incrementan consistentemente según el tamaño de array incrementa, de forma proporcional y casi lineal, mientras que en el lanzado en GPU a partir de los 400.000 no experimenta gran variación, sino que parece mantenerse relativamente estable

Podemos ver una gráfica de speedup de los dos programas y hacernos una idea del orden de magnitud en el que los tiempos mejoran:



Esto ocurre así porque la CPU no está pensado para realizar tareas de paralelización masivas, como si está preparada la GPU, ya que esta cuenta con miles de cores (a menos frecuencia, eso sí) que una CPU, por lo que la paralización del programa es mucho mejor y obteniendo unos resultados mucho mejores como ya hemos podido observar.

* + What BLOCK SIZE (number of threads per block) have you used? Do you think it is the most optimal? Explain.

El valor de la variable BLOCK SIZE, define la cantidad de hilos por bloque. Para que sea lo más óptimo posible, el valor de esa variable ha de ser múltiplo de 32. Ya que un bloque de threads consiste de 32-thread warps. En nuestro caso hemos utilizado BLOCK SIZE = 16. De esta forma, los bloques ocupan siempre todos los posibles huecos posibles, permitiendo aprovechar al máximo los recursos.

* Exercise 2:
  + Fill in a table with time and speedup results compared to your manually vectorized CPU code for images of different resolutions (SD, HD, FHD, UHD-4k, UHD-8k). You must include a column with the fps at which the program would process. Discuss the results.

| Calidad | Version GPU | | Version Auto-Vec | | SpeedUp |
| --- | --- | --- | --- | --- | --- |
| Tiempo | FPS | Tiempo | FPS |
| SD | 0.000021 | 47619.04 | 0.000192 | 5208.33 | 9.1428 |
| HD | 0.000022 | 45454.54 | 0.000538 | 1858.73 | 24.4545 |
| FHD | 0.000022 | 45454.54 | 0.000926 | 1079.91 | 42.0909 |
| 4k | 0.000024 | 41666.66 | 0.003624 | 306.37 | 151.0000 |
| 8k | 0.000026 | 38461.53 | 0.012740 | 78.49 | 490.0000 |

En la tabla anterior se encuentran los tiempos de ejecución de realizar las conversiones a gris de una imagen, para distintas resoluciones de la misma. El primer conjunto de datos procede de la implementación pedida de GPU. El segundo conjunto, pertenece a la implementación del mismo programa, solo que vectorizado, realizado en la práctica anterior.

Como se puede apreciar, los tiempos del programa GPU son muchos más rápidos. Siendo entre 9 y 490 veces más rápido, dependiendo de la resolución de la imagen. Está diferencia va haciéndose más significativa a cuanta mayor resolución tenga la imagen. Esto se debe a que cuanta mayor calidad, mayor tamaño, lo que se traduce en mayor tiempo de cómputo. Por lo que mayor importancia tiene la paralelización con “cuda”.

* + Explain how you implement the algorithm to be optimal for GPU

Para que el algoritmo sea óptimo para ser ejecutado en la GPU, debemos implementar el algoritmo siguiendo las siguientes indicaciones:

1. Función getRGB (Kernel de CUDA): Este kernel se encarga de calcular la conversión de RGB a escala de grises en paralelo. Cada hilo del kernel procesa un píxel de la imagen y realiza la conversión. El kernel toma un puntero a la imagen en formato RGB, el ancho y alto de la imagen, la cantidad de canales y un puntero a la imagen en escala de grises.
   * El cálculo de la escala de grises se basa en la fórmula estándar de conversión, que toma en cuenta la contribución de los canales RGB a la intensidad de grises.
   * blockIdx.x y threadIdx.x se utilizan para determinar la posición del hilo en el bloque y el bloque en la cuadrícula.
2. Función main:
   * El programa comienza cargando una imagen RGB desde un archivo utilizando la biblioteca stb\_image. Luego, se reserva memoria tanto en el host como en el dispositivo (GPU) para las imágenes de entrada y salida.
   * Se copia la imagen RGB desde la memoria del host al dispositivo utilizando cudaMemcpy.
   * Se crea un nombre de archivo para la imagen en escala de grises y se realiza la conversión de RGB a escala de grises en la GPU utilizando el kernel getRGB lanzando un thread por cada píxel, de esta forma, cada thread queda encargado de realizar la conversión a grises de su píxel y al no haber dependencias no debemos preocuparnos de la condición de carrera o dependencias entre ellos.
   * Se copia la imagen en escala de grises desde la GPU al host.
   * Se guarda la imagen en escala de grises en un archivo utilizando la biblioteca stb\_image\_write.