**Integrantes:** Javier Fraile Iglesias e Iván Fernández París. Grupo 1472. Pareja 04

**ASIGNATURA Computación de altas prestaciones**

**Task 2**

**GPU Programming**

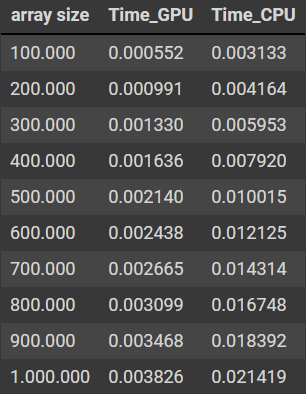
You must write a report answering the questions proposed in each exercise, plus the requested files. Submit a zip file through Moodle. Check submission date in Moodle (deadline is until 11:59 pm of that date).

**DESTACAR:** Dejamos entregados junto a esta memoria los notebooks con la implementación que corresponde con cada ejercicio en dos carpetas con el nombre de los ejercicios.

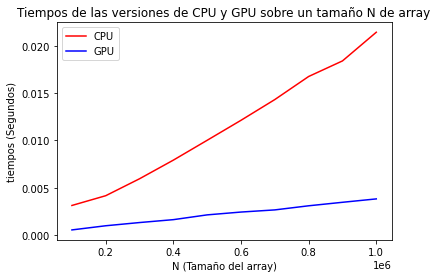
* **Exercise 1:**

Junto a la memoria entregamos una carpeta llamada “/Ej1” en la que se encuentra un archivo .ipynb, donde está la implementación tanto de la versión CPU 1D stencil, como de GPU 1D stencil en celdas independiente y en otra celda ambas versiones a la vez.

* + **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.**

Tras la ejecución para valores desde 100.000 hasta 1.000.000, los resultados obtenidos son los siguientes (destacar que los tiempos son los resultados de una única ejecución, sería más conveniente ejecutarlo varias veces y sacar la media):

A partir de los tiempos obtenidos (reflejados en la tabla anterior) trazamos una gráfica donde en el eje X se refleja el tamaño que tendrá el array (valor de N) y en el eje Y el tiempo, quedando la gráfica tal que así:



Respecto a los resultados obtenidos, podemos observar que cuanto más grande es el tamaño del array, más es la diferencia de tiempo entre la ejecución de la versión GPU 1D stencil frente a la versión CPU 1D stencil.

Esto se debe a que el flujo de ejecución del código que se ejecuta en el device, es decir, aquella función declarada como \_\_global\_\_ que se ejecuta en paralelo (aunque luego se haga una sincronización), mientras que en el caso de la ejecución de una función que se ejecuta en el host se ejecuta en serie.

Cuando el tamaño del array es muy pequeño, por mucho que se paralelicen las acciones sobre el array (en el caso de este ejemplo), el flujo de procesamiento conlleva no solo la ejecución de la función \_\_global\_\_ en el device, si no también copiar los datos de la memoria de la CPU en la de la GPU, ejecutar el código en el device y volver a copiar la memoria de la GPU en la memoria de la CPU. Esto supone un coste temporal añadido que la versión CPU no tiene en cuenta ya que no necesita copiar nada de una memoria a otra, por lo que si el tamaño del array es pequeño, el coste de tener que estar copiando memoria puede afectar negativamente. Pero cuando el tamaño del array es muy grande, compensa tener que copiar la memoria de la CPU en la de la GPU puesto que luego las operaciones sobre el array se harán en paralelo y tardará mucho menos que realizarlo en serie como ocurre en la versión de CPU o host.

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

El número de hilos por bloque o BLOCK SIZE utilizado ha sido de 40.

Como bien se comenta en las diapositivas, los bloques de hilos se ejecutan en los Streaming Multiprocessors (SMs). Es por ello que consideramos que para la ejecución de 1D stencil usar **40** **como tamaño de bloque es el más óptimo** ya que al ser el mismo número de Streaming Multiprocessor con los que cuenta la gráfica se maximizará su rendimiento.

Aclaraciones: según hemos visto, el HW que nos ofrece Google Colab cuenta con una Tesla T4 y buscando las especificaciones de dicha práctica, hemos encontrado la cantidad de SMs con los que cuenta (Link a la referencia [Nvidia Tesla T4 Specs - GadgetVersus](https://gadgetversus.com/graphics-card/nvidia-tesla-t4-specs/) ) y de ahí determinamos el valor del BLOCK SIZE.

* **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.**

A continuación se adjuntan tres tablas, la primera hará referencia a los tiempos obtenidos con la versión “intrinsic”, la segunda con la versión auto-vectorizada y la tercera a la versión de la GPU. Los datos se han obtenido ejecutando una vez la conversión a gris de cada imagen para cada una de las versiones, pero sí que sería más correcto realizar varias veces la ejecución y calcular una media puesto que en cada ejecución los tiempos varían.

En la tabla de GPU mostramos tanto la aceleración respecto a la versión “intrinsic” y a la auto-vectorizada para así ver como mejora en comparación con ambas versiones.

* **Intrinsic:**

| **Resolución** | **Tiempo (s)** | **SpeedUp** | **Tasa de FPS (1/s)** |
| --- | --- | --- | --- |
| **SD** | 0.004042 | 1 | 247.40 |
| **HD** | 0.013360 | 1 | 74.85 |
| **FHD** | 0.033521 | 1 | 29.83 |
| **4k** | 0.128500 | 1 | 7.78 |
| **8k** | 0.535656 | 1 | 1.87 |

* **Auto-vectorizada:**

| **Resolución** | **Tiempo (s)** | **SpeedUp** | **Tasa de FPS (1/s)** |
| --- | --- | --- | --- |
| **SD** | 0.003159 | 1.189300412 | 316.5558721 |
| **HD** | 0.010500 | 1.444095238 | 95.23809524 |
| **FHD** | 0.021804 | 1.44267107 | 45.86314438 |
| **4k** | 0.077053 | 4.028733469 | 12.97808002 |
| **8k** | 0.296005 | 3.675741964 | 3.378321312 |

* **GPU:**

| **Resolución** | **Tiempo (s)** | **SpeedUp en comparación con intrinsic** | **SpeedUp en comparación con auto-vectorizada** | **Tasa de FPS (1/s)** |
| --- | --- | --- | --- | --- |
| **SD** | 0.000176 | 22.97 | 17.95 | 5681.82 |
| **HD** | 0.000639 | 20.91 | 16.43 | 1564.95 |
| **FHD** | 0.001357 | 24.70 | 16.07 | 736.92 |
| **4k** | 0.005208 | 24.67 | 14.80 | 192.01 |
| **8k** | 0.020260 | 26.44 | 14.61 | 49.36 |

* La aceleración se ha calculado de la siguiente forma: S = Toriginal / T
* La tasa de FPS se ha calculado de la siguiente manera: FPS = 1 / T
* El hace Toriginal referencia al tiempo obtenido en la versión “intrinsic” para así poder ver qué mejora se obtiene
* **Discusión:**

Como se puede observar la ejecución en GPU es unas 22 veces más rápida que la versión “intrinsic” y unas 16 veces más rápido que la versión auto-vectorizada de la CPU.

Esto se debe a que en la versión de la GPU los píxeles se modifican a la vez al lanzar un número de hilos igual a los píxeles que tenga la imagen, mientras que para las versiones de la CPU debe realizar varias vueltas de bucle.

Destacar que el tiempo que se contabiliza para la versión “intrinsic” y en la auto-vectorizada corresponde con llamar a la función RGB durante todas las iteraciones que sean necesarias, mientras que en la versión de la GPU se contabiliza el tiempo que se tarda en copiar memoria del host al device, llamar al kernel y copiar la memoria del device al host.

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

En primer lugar, declaramos las device copies que se van a utilizar en el kernel, siendo estas, la imagen original, el resultado de la imagen en gray, el ancho, el alto de la imagen y el número de canales (que siempre será 4).

A continuación reservamos memoria para dichas device copies con cudaMalloc, y copiamos el valor de memoria de las variables del host a las del device.

Declaramos las variables dim3 para poder establecer las variables de dos dimensiones. De esta manera, ahora podremos trabajar en el device con arrays 2D. En dichas variables lo que se indica es el número de blocks per grid (nblocks) y el número de threads per block (threadsPerBlock) de modo que lancemos 1 hilo por cada píxel con el que cuenta la imagen.

Una vez hecho esto llamamos al kernel de la siguiente manera:

pixelOperations<<<nblocks, threadsPerBlock>>>(d\_rgb\_image, d\_grey\_image, d\_width, d\_height, d\_nchannels);

En el kernel, calculamos a qué fila y columna (píxel) corresponde el hilo de la siguiente manera:

int ROW = blockIdx.y\*blockDim.y+threadIdx.y;

int COL = blockIdx.x\*blockDim.x+threadIdx.x;

Seguidamente, comprobamos que dicha fila y columna no se salgan de los límites de la foto, lo que significa que el valor de ROW no sea superior al alto (height) de la imagen y que el valor de la COL no sea superior al del ancho (width) de la imagen (ambas condiciones se deben de cumplir a la vez para poder cambiar el color al pixel), y en caso de no salirse se cambia el color del pixel a gray y guardamos el valor del nuevo píxel.

Y una vez que ha terminado la ejecución del kernel se copia la memoria del device al host.