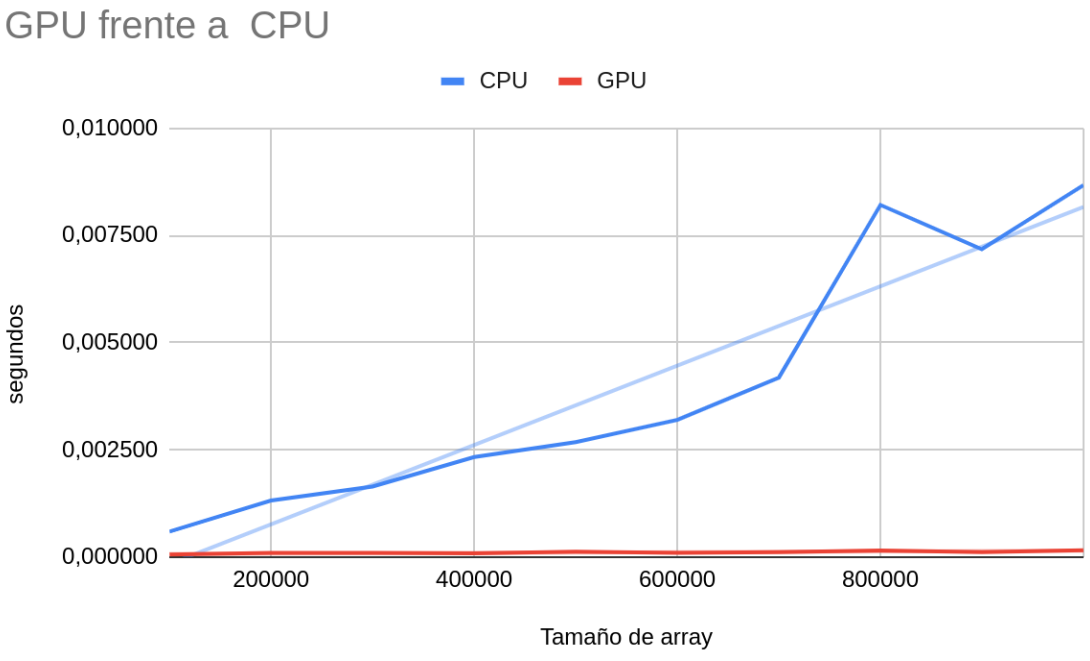
ASIGNATURA Computación de altas prestaciones

Task 2
GPU Programming

- 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.

Reps	CPU	GPU
100000	0,000593	0,000058
200000	0,001319	0,000096
300000	0,001645	0,000100
400000	0,002332	0,000085
500000	0,002680	0,000126
600000	0,003197	0,000099
700000	0,004185	0,000116
800000	0,008211	0,000152
900000	0,007177	0,000114
1000000	0,008671	0,000162



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

Hemos usado un block Size de 256.

Trás una pequeña búsqueda en internet, vimos que un valor típico para el ordenador que se nos había asignado en el collab podría ser de 128, sin embargo, vimos que el proceso de elección del block size tiene un factor de experimentación y que es importante no pasarse de tamaño. Probamos algún valor más y el que mejores resultados nos aportó fue el de 256, por eso, finalmente decidimos fijar ese valor.

- 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.

	CPU	GPU	SpeedUp	FPS (GPU)
HD	0,065777	0,065107	1,010290752	15,35933156
SD	0,021438	0,063678	0,3366625836	15,7040108
FHD	0,128515	0,141072	0,910988715	7,088578882
4K	0,476513	0,543069	0,8774446709	1,841386638
8K	1,780921	1,390569	1,280713866	0,7191300827

Tras sacar todos los datos vemos que el rendimiento es algo mejor en la GPU que en la CPU, pero tampoco supone un cambio muy notable. Los datos no son muy buenos para haber estado usando la GPU, pero los tiempos usando google collab estaban dando altos en general, no solo en GPU si no en CPU también, por lo que ver que hay un poco de mejoría es lo importante, más que comparar el tiempo con los de la práctica anterior.

- Explain how you implement the algorithm to be optimal for GPU

Para optimizar el código del programa de GPU hay varias cosas que podríamos mirar, como paralelizar los cálculos de la GPU o intentar minimizar la transferencia de datos entre la CPU y la GPU. Al igual que utilizar la memoria compartida para almacenar datos temporales cuando sea posible.

```
__global__ void multiplyRatios(uint8_t *in, float *out, float* d_in_mul) {
    __shared__ uint8_t aux[BLOCK_SIZE];
    int gindex = threadIdx.x + blockIdx.x * blockDim.x;
    int lindex = threadIdx.x;

    aux[lindex] = in[gindex] * d_in_mul[lindex];

    __syncthreads();

    out[gindex] = aux[lindex];
}

__global__ void rgbToGrey(float *in, uint8_t *out) {
  int gindex = threadIdx.x + blockIdx.x * blockDim.x;
  int gindex_temp = gindex * 4;
  int result = 0;
  for (int offset = 0 ; offset < 4 ; offset++) {
    result += in[gindex_temp + offset];
  }

  out[gindex] = result;
}

```

Aquí llamamos a las funciones de kernel para GPU:

```
cudaMemcpy(d_in_mul, mul_ratios, sizeof(mul_ratios), cudaMemcpyHostToDevice);
multiplyRatios<<<(size*4)/BLOCK_SIZE, BLOCK_SIZE>>>(d_in, d_out_mul, d_in_mul);
rgbToGrey<<<size/BLOCK_SIZE, BLOCK_SIZE>>>(d_out_mul, d_out);
cudaMemcpy(grey_image, d_out, size * sizeof(uint8_t), cudaMemcpyDeviceToHost);
stbi_write_jpg(grey_image_filename, width, height, 1, grey_image, 10);
```