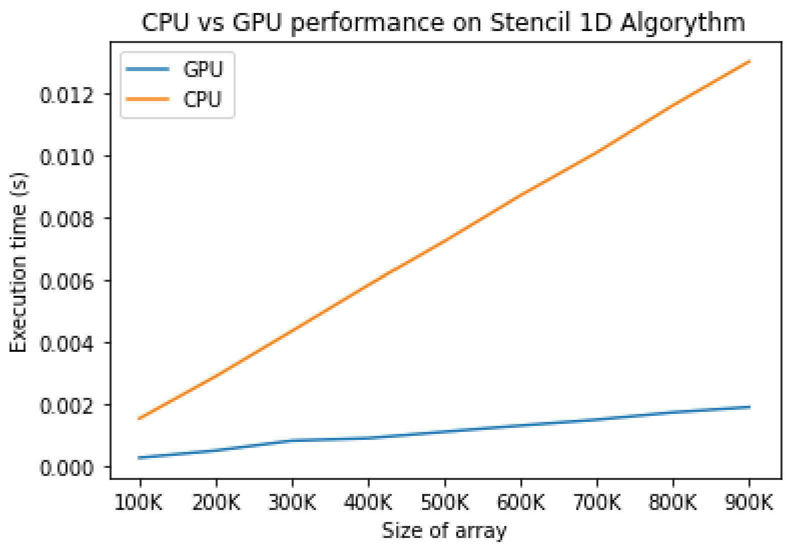
ASIGNATURA Computación de altas prestaciones

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).

- 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.

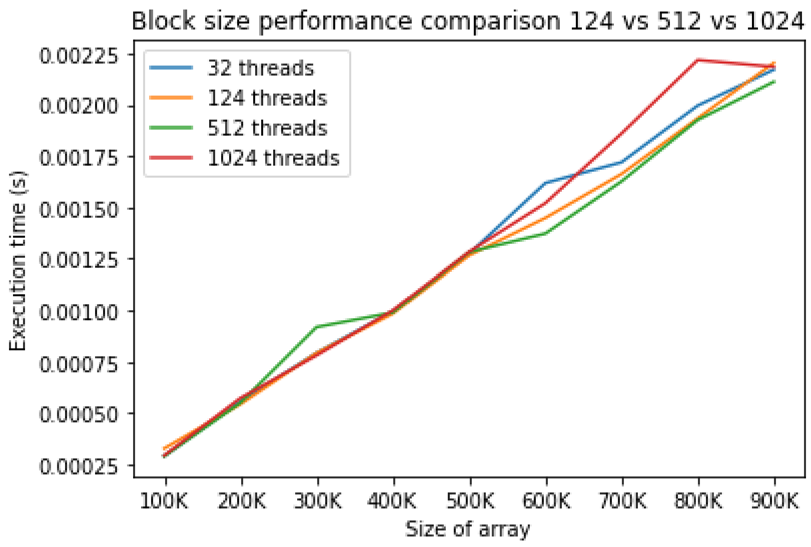
    We developed a version of the stencil 1d algorithm to address parallel vectorization using both blocks and threads. Then we developed a CPU version of the same algorithm and compared the results.



As we can see, the GPU performance is much higher than the CPU one. This is due to the stencil 1d algorithm using each of the elements of the array multiple times. The GPU program uses threads with shared memory, allowing us to access fewer times each of the necessary elements of the array. This approach results in a faster execution time.

o What BLOCK SIZE (number of threads per block) have you used? Do you think it is the most optimal? Explain.
Block size represents the amount of threads each block is divided into. Threads can work with shared memory to share data between different calculations, resulting in a faster parallelization.

Our first hypothesis to understand the best block size is that the bigger the block size is, the more sharing will be done between threads, resulting in a better performance. We tested this hypothesis with the following results:



As we can observe in this plot, the expected improvement in performance cannot be noticed. This could be due to various reasons, including a bottleneck in shared memory capabilities restricting the improvement in performance. Another factor that can be responsible for the results is the size of the data, where if the sample is not big enough the growth difference would not be accurate.
However, this improvement will end whenever the block size becomes to big and the bottleneck of the maximum amount of threads the GPU can handle becomes a problem.

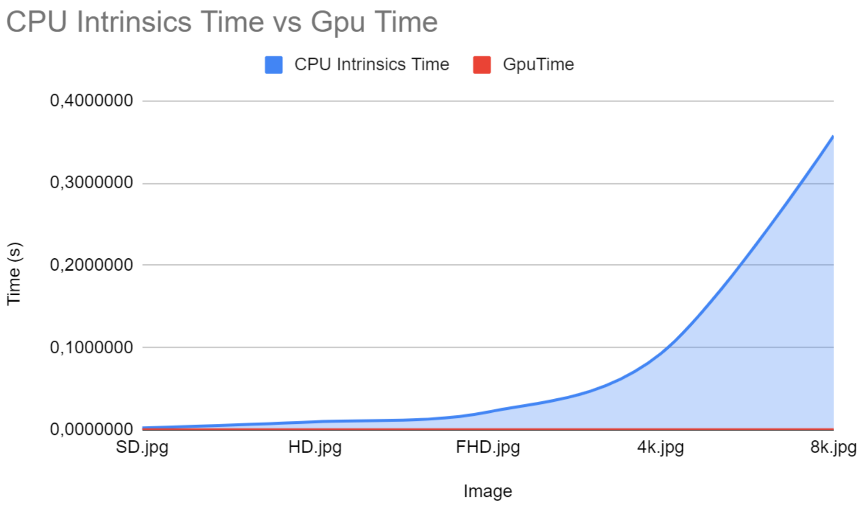
The way to achieve the best performance possible is to execute the biggest amount of threads possible that our GPU can handle. This may change depending on the hardware used.

- 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.

      Using the cuda tools we created a new version of the gray image conversion algorithm sending all the computation parts to the device (GPU). This version shows the following results:

Image	CPU Intrinsics Time	GpuTime	Speedup	FPS GPU
SD.jpg	0,0024510	0,0000260	94,26923077	38461,53846
HD.jpg	0,0097980	0,0000270	362,8888889	37037,03704
FHD.jpg	0,0217060	0,0000300	723,5333333	33333,33333
4k.jpg	0,0927180	0,0000280	3311,357143	35714,28571
8k.jpg	0,3575180	0,0000310	11532,83871	32258,06452

As we can observe, the results show really high speedup and fps count. This is due to only measuring the processing effort. If other parts of the program were taken into account such as the image and file creation, both GPU and CPU times would be increased in a constant and significant way resulting in a slower speedup.



As we can observe in the results, the images obtain a slower result using the CPU program than when using the GPU one. The bigger images are processed much faster with the GPU version. To justify this results, we developed the following hypothesis:

When using the GPU is much faster as it uses multiple threads sharing data, needing less memory access and a faster result.

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

  To implement an optimal algorithm for GPU we must delegate all computation effort to the device, and balance block size to achieve the fastest result possible. This was done by copying the original image array to CUDA format to be able to work with it with the GPU, and separating it in blocks and strings determined by the maximum amount of strings a block can handle, in our case 1204, and the amount of blocks necessary to achieve the thread objective. This will result in the device parallelizing rows of the image and sharing resources between the threads to save computation effort.