Part 1:

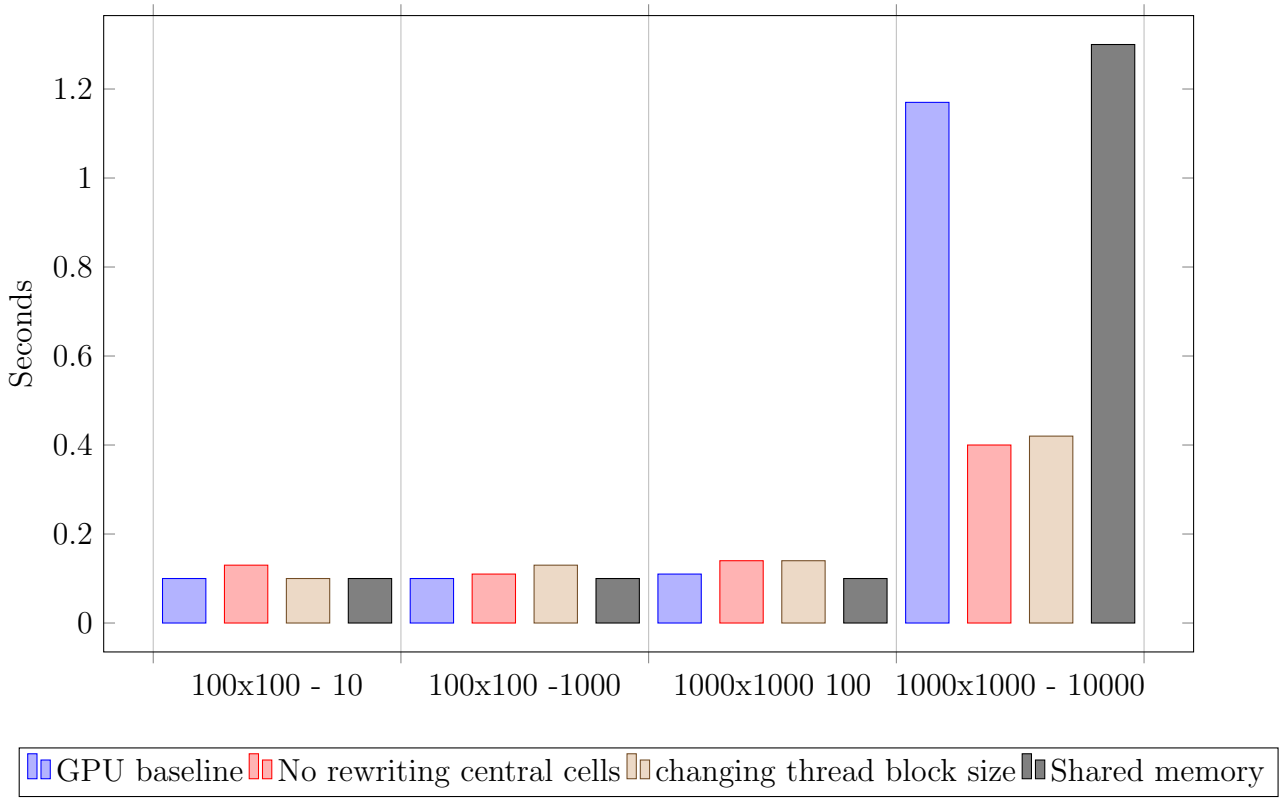
To implement *GPU_array_process*, we first allocate some array in gpu for the input and the output and we copy the input array from CPU to the GPU memory. Then we define the settings of the block's (size and disposition). Then call a function that compute one iteration in a loop in order to compute the right number of iterations. At the end, we copy the output to the CPU output array and free the allocated arrays. There's 3 function defined that perform computation for 1 iteration. The first one is simple and implement no optimization. The second function don't compute any of the center element since they doesn't change over time. The third function implement shared memory. We also tried different setup of block size and disposition.

Part 2:

Version	Computation time	Speed up
simple GPU	1.067s	1
Best block setting	0.4268s	2,5
shared memory	1.351	0.79
avoiding center cells	0.403	2.65

We can see that choosing the right block's setting (size and position) increase a lot the performances. Avoiding the computation of center cell also improve performance a lot. The shared memory implementation doesn't work as expected, this is probably can come from two reasons, either we didn't manage accesses bank well, or this can come from the extensive use of L1 cache in the simple implementation. In fact the simple version of the algorithm is using relatively small-sized thread blocks which can benefit form L1 cache that are shared among all threads in a block. Because of the small size the used part of the data can fit the cache and the execution does not encounter as many cache miss as we can think and the value we compute is pretty straightforward so it does not take a long time to compute. Hence the cost of copying data to the shared memory is not worth it.

Here are the results we obtained for each one of the previously optimisations:

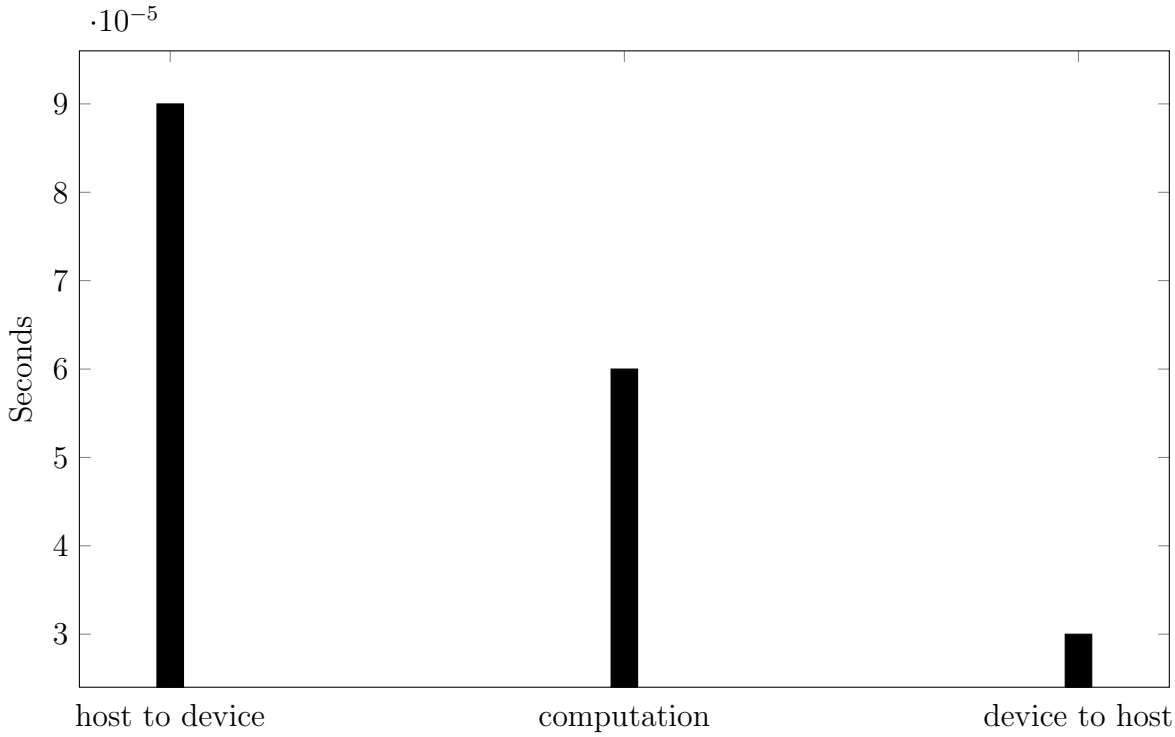


Part 3:

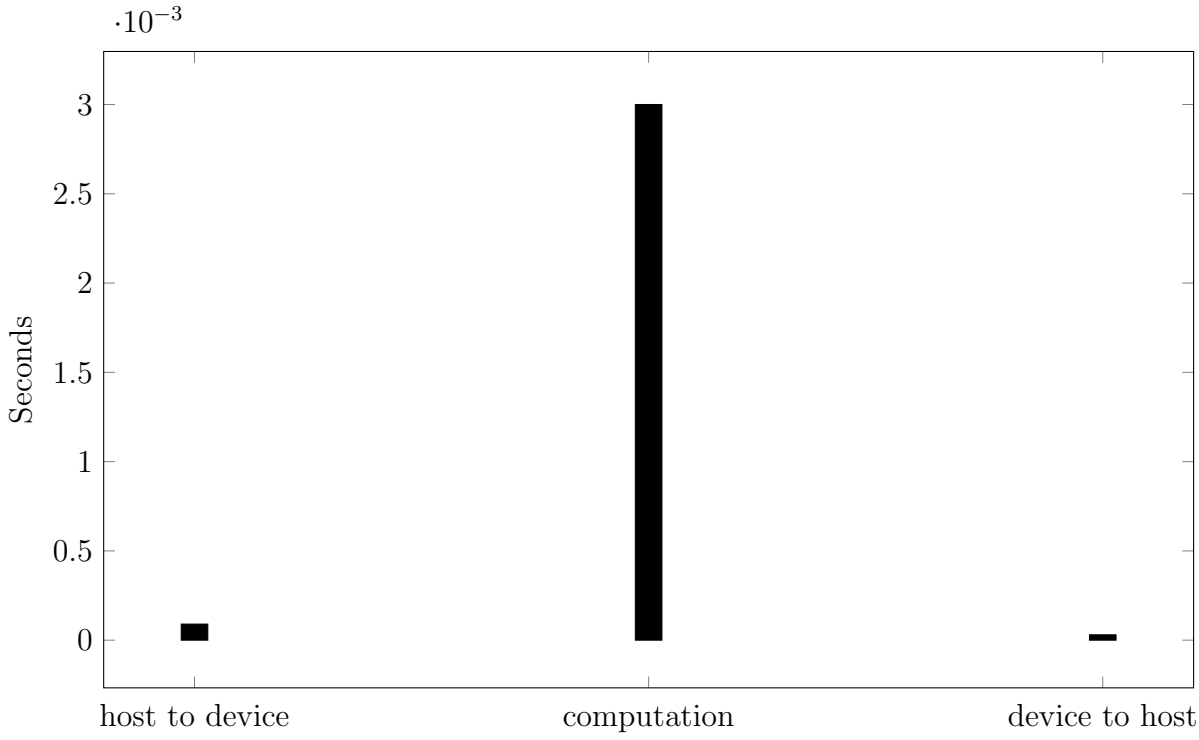
	100, 10	100, 1000	1000, 100	1000, 10000
CPU	0.001	0.025	0.16	15.456
GPU	0.1	0.1	0. 1	0.4

For CPU vs GPU we decide to go for a table instead of graph since the difference is way to big between the run 1000x1000 with 10K iterations to see something interesting on a graph.

GPU runtime breakdown 100x100 - 10

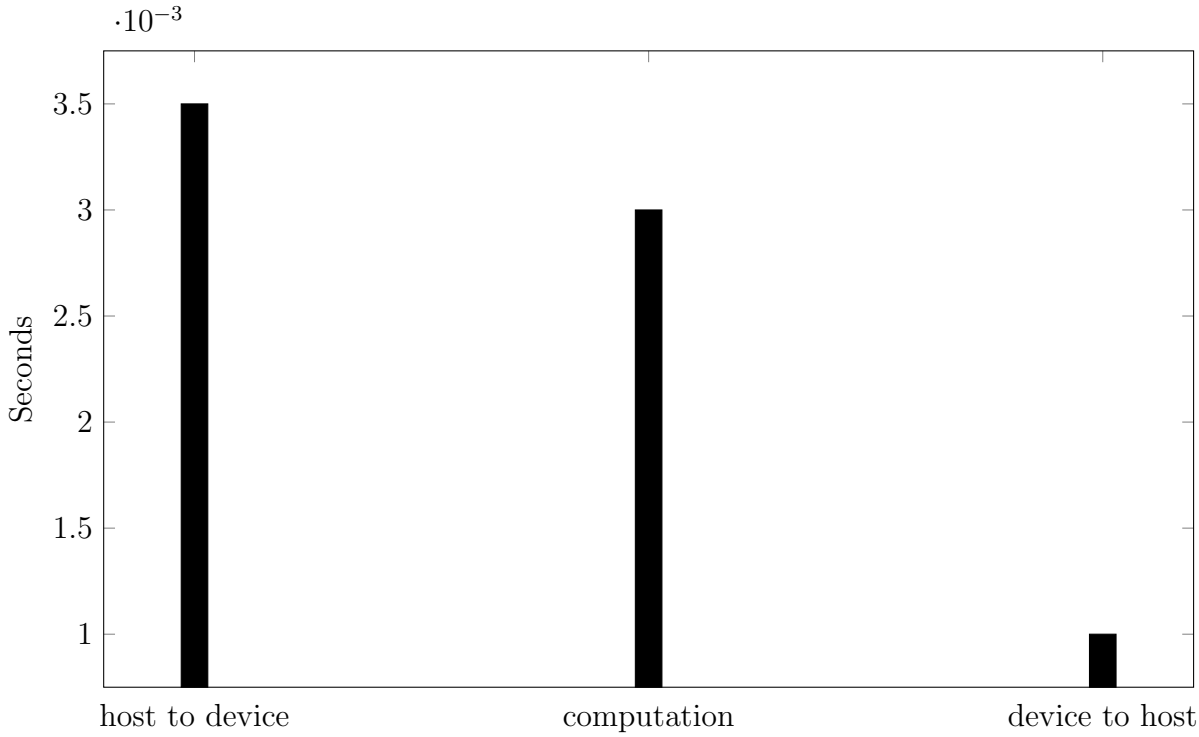


GPU runtime breakdown 100x100 - 1000

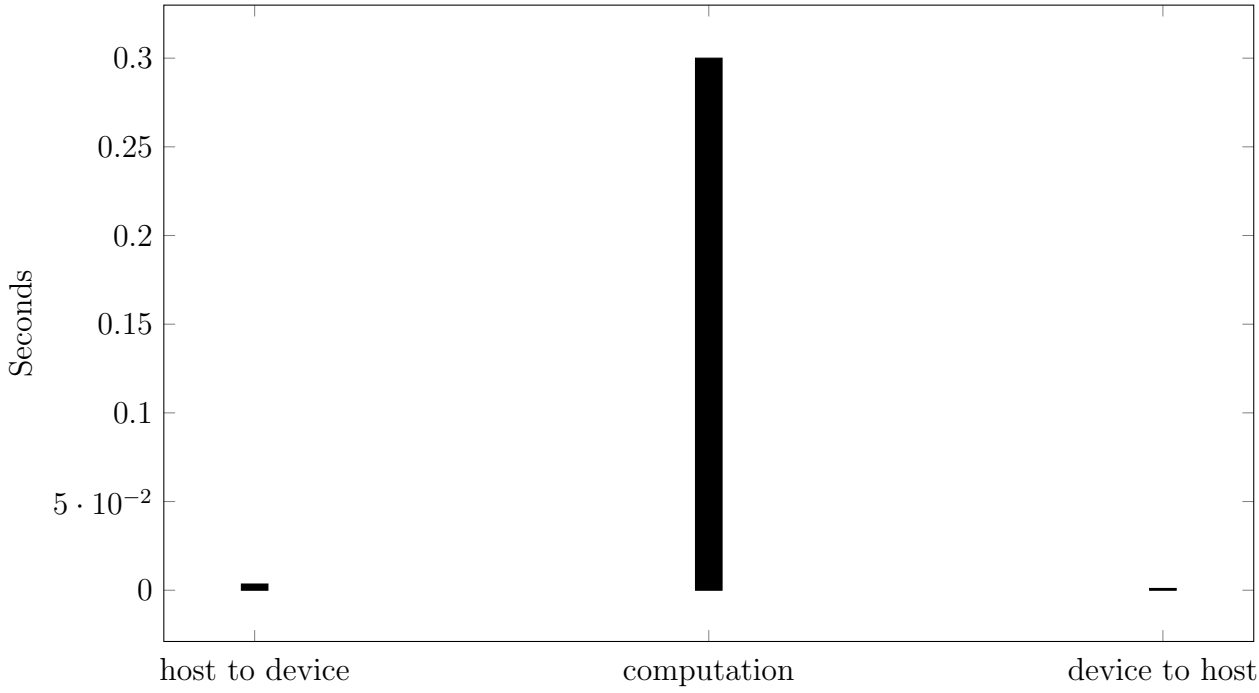


GPU runtime breakdown 1000x1000 - 100

Assignement 4



GPU runtime breakdown 1000x1000 - 10000



Part 4:

GPU allows us to get a speedup of ... over CPU implementation which is quite good. The time spend in copying data from/to host vary depending of the size of the array, for array of small size, it's not interesting to use CPU since those copies take a lot of time. The GPU is legitimate on very big input and large iteration.