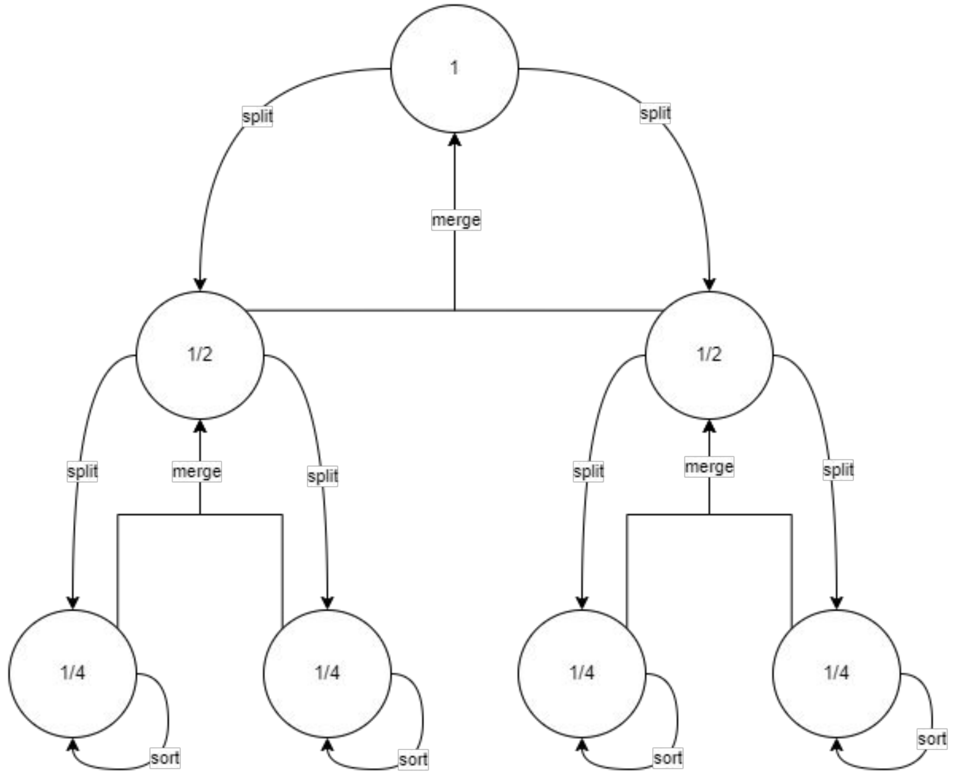
CUDA &
Integers Sorting

CLE3_T3G1 - Guilherme Antunes (103600) - Pedro Rasinhas (103541)

Prof. António Rui de Oliveira e Silva Borges

Integers Sorting



- Call the global function in host
- Split array till we have K sub arrays
- Sort each of the subarrays in each thread
- Merge two by two till we have 1 again with half of the previous threads
- Wait for thread synchronization in each round and update its quantity
- Validate the result

Integers Sorting - Results (dataSeq1M.bin)

Cuda Threads	Row (Avg Time)	Col (Avg Time)
1	6.159 s	17.360 s
2	6.159 s	17.174 s
4	3.984 s	10.633 s
8	2.999 s	7.613 s
16	2.483 s	6.178 s
32	2.277 s	5.929 s
64	2.160 s	5.460 s
128	2.116 s	5.223 s
256	2.107 s	5.131 s
512	2.108 s	5.088 s
1014	2.113 s	5.0757 s

*Average of 5 runs;

Integers Sorting - Results (dataSeq1M.bin)

Quantity	1	2	4	8
CPU Threads				
Avg Time (s)*	0.447	0.252	0.192	0.144
Processes				
Avg Time (s)*	0.394	0.241	0.186	0.200
CUDA Threads				
Avg Time (s)*	6.159	6.159	3.984	2.999

Integers Sorting - Conclusions

- GPU, for this particular exercise, is NOT worth it, as
- In all the iterations, all the integers from the array are used. However, the cache can't fit all the integers, therefore the global memory is accessed frequently, and this process takes much more time than accessing the cache.
- Using rows instead of columns is faster, due to the extra calculations done in the column version;
- Until 32 threads we can see that the time is decreasing, as expected, and after that it starts stabilizing, which is explained by the fact that only 32 threads can execute at once in a SIMD block,