universidade
de aveiro

Compute Unified Device Architecture

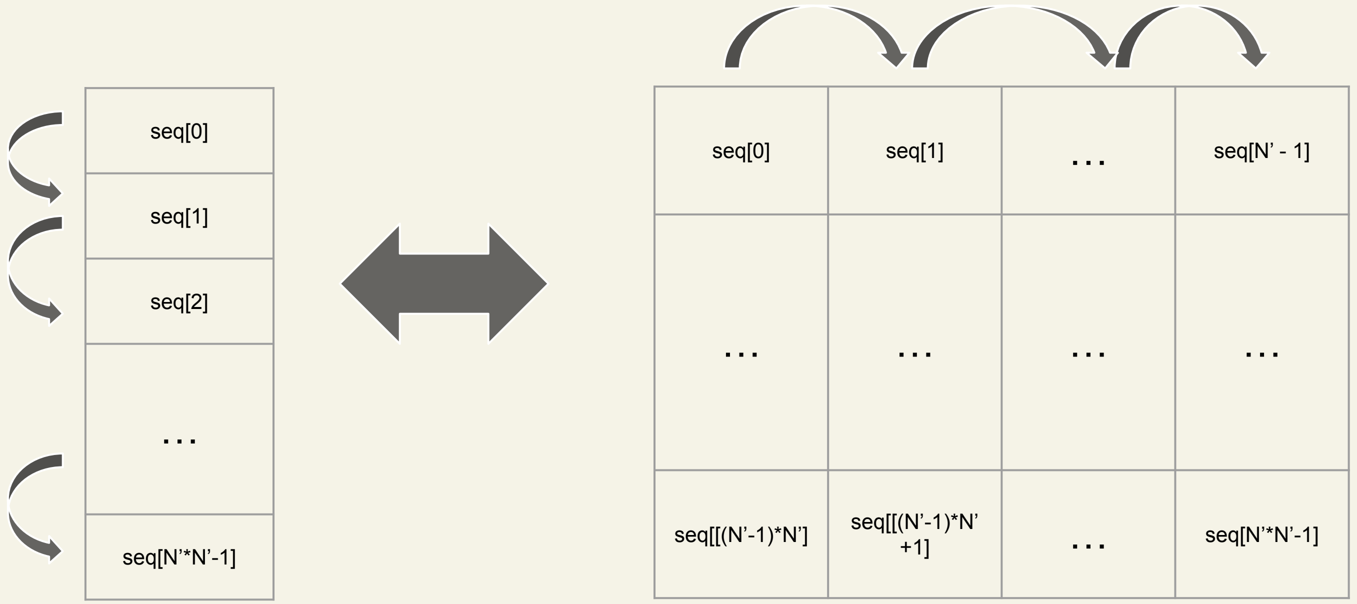
Computação em Larga Escala - Assignment III

Diogo Magalhães	102470
Rafael Gil	118377

Problem 1 - Implementation

Objective: Perform bitonic sort, by processing the rows.

numArray[(N'/K)*(1<<iter)*idx+1]

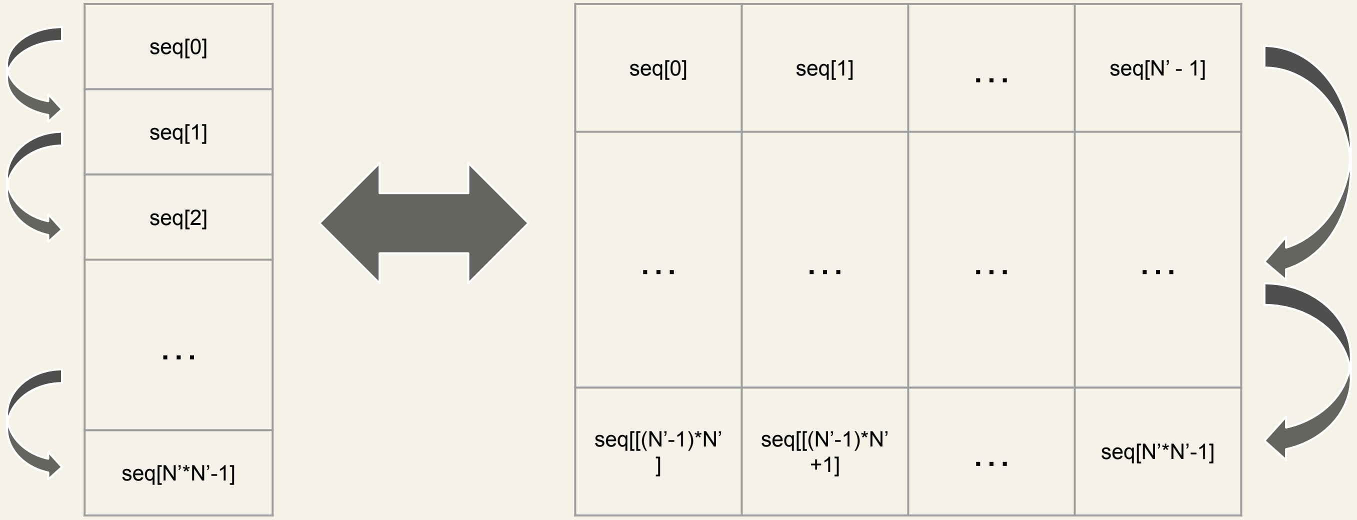


Threads	Time(s)
1	9.648
2	5.736
4	3.904
8	3.037
16	2.638
32	2.480
64	2.378
128	2.341
256	2.340
512	2.347
1024	2.351

Problem 2 - Implementation

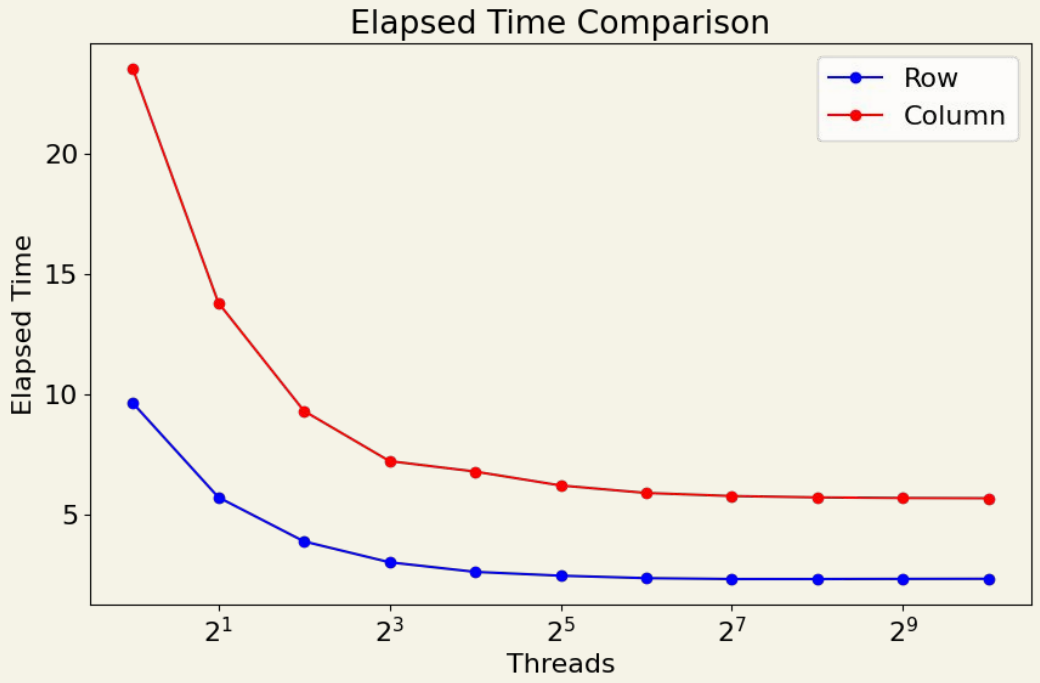
Objective: Perform bitonic sort, by processing the columns.

numArray[(N'/K)*(1 << iter) * idx + N' *(i mod N') + (i div N')]



Threads	Time(s)
1	23.536
2	13.813
4	9.322
8	7.239
16	6.805
32	6.227
64	5.916
128	5.789
256	5.731
512	5.707
1024	5.696

Row vs Column processing

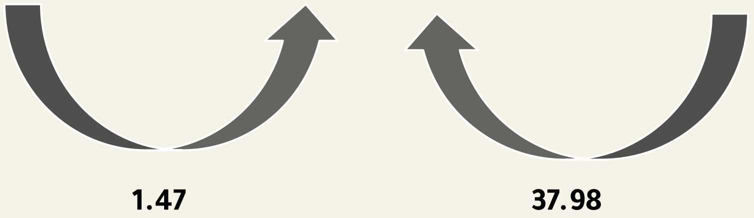


Threads	Row Speed Up	Column Speed Up
1	1.000	1.000
2	1.682	1.704
4	1.469	1.481
8	1.285	1.288
16	1.151	1.064
32	1.067	1.093
64	1.043	1.053
128	1.016	1.022
256	1.001	1.010
512	0.997	1.004
1024	0.998	1.002

Previous results

Best times obtained using the different methods

	Multi-threaded	MPI	CUDA
Time (s)	0.0907	0.0616	2.34



Conclusions

GPU and CPU are two different types of hardware that can be used to execute code. The main difference between the two is that GPU is designed to handle parallel tasks, having a larger number of cores, while CPU is designed to handle sequential tasks, having higher clock speeds. GPU cores have less cache memory and simpler instruction sets, which makes them less powerful than CPU cores.
When we analyse the previous results, we can conclude that:

- Even though the GPU has a lot of cores, the performance improvement is not as high as expected. By doubling the number of threads, the time does not reduce to half, and sometimes it even increases due to the need for constant synchronization of threads.
- The GPU is more efficient when we use row ordering. The higher complexity of the column ordering operations, when compared to row ordering, and the cache optimizations that the GPU is capable of doing when using row ordering, make the GPU more efficient.
- The multiprocessing and multithreading CPU implementations ended up being way more performant than the GPU implementation. In our tests we were able to get a speed up of 37.98 from the CUDA implementation to the MPI implementation.

Is it worthwhile to use the GPU to solve this kind of problem?
For this specific problem, the GPU is not the best solution. The large number of threads being created and their need for constant synchronization adds an unnecessary overhead to the problem time performance. For problems without any interdependencies, a GPU is capable of providing a significant performance improvement.

Problem 1 - Conclusions

When we analyse the previous tables, we can conclude that:

- In the first table, it is possible to see the impact of using smaller and bigger buffer sizes. Bigger buffer sizes, usually means less buffer requests from the worker to the dispatcher, so the worker will be less times stopped waiting for the dispatcher, meaning, usually, faster times.
  In our tests, where we changed the buffer size from 1024 to 8192, we got improvements of 9.33% when using 1 worker, 17.15% when using 2 workers, 8.20% when using 4 workers and 8.25% when using 8 workers.
- In the second table, it is observed that depending on the size of the file, the usage of more processes might be prejudicial to the performance of the program since for smaller files processes might be created but not used since all the file would have already been processed by another process.
- Overall, the results obtained and shown in both tables were expected and show that the performance of the text processing improved a lot with the usage of MPI.
  The improvements between analyzing all the files and using a fixed 8192 buffer size where of 38.16% from 1 to 2 workers, 32% from 2 to 4 workers and 13.5% from using 4 to 8 workers, which means an improvement of 63.62% from 1 to 8 workers.