Sorting Sequences of Values

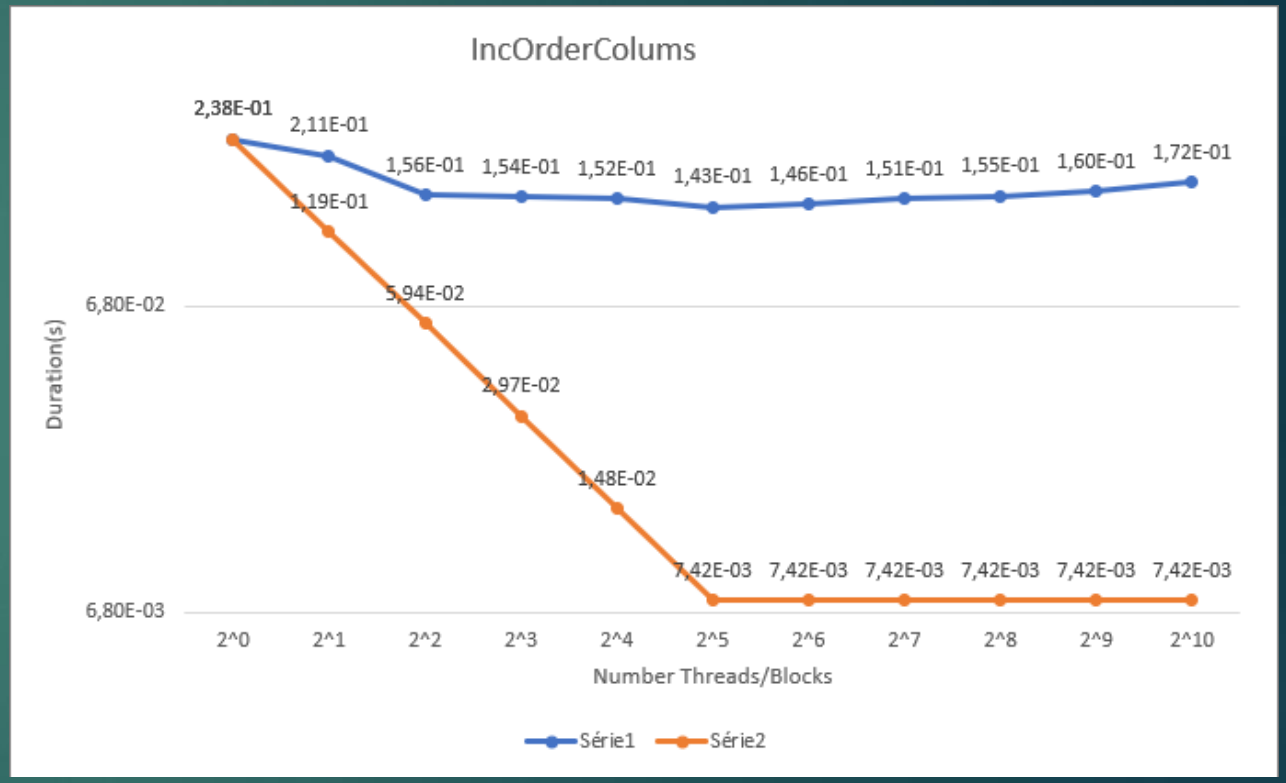
GPU CUDA OPTIMIZATION

universidade de aveiro

João Torrinhas nº98435
Diogo Torrinhas nº98440

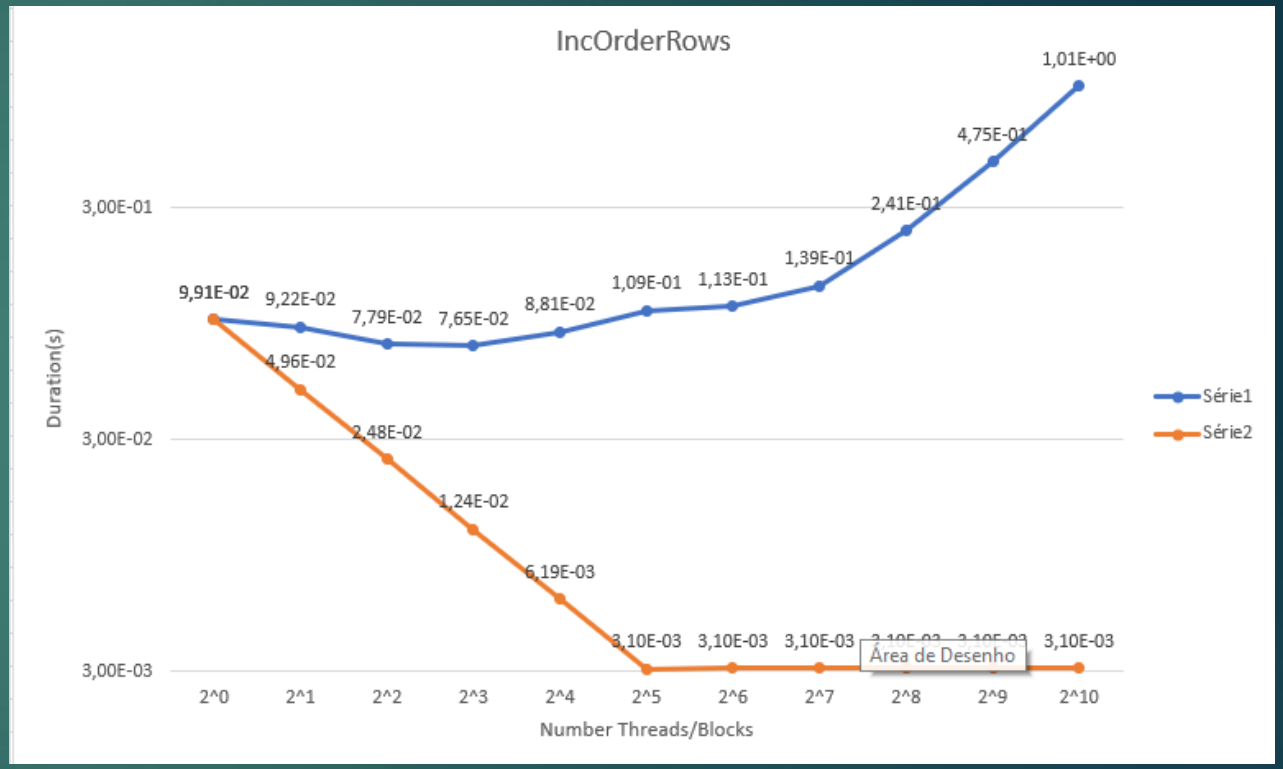
Launch configuration for Colums X

- Os melhor resultado foi o 0,14272 ,<<(32,1,1) , (32,1,1)>>
- Melhor valor para:
  - blockDimX = 1 << 5
  - blockDimY = 1 << 0
  - gridDimX = 1 << 5
  - gridDimY = 1 << 0



Launch configuration for Rows X

- Os melhor resultado foi 0,07649 , <<(128,1,1) , (8,1,1)>>
- Melhor valor para:
  - blockDimX = 1 << 3
  - blockDimY = 1 << 0
  - blockDimZ = 1 << 0
  - gridDimX = 1 << 7



Optimal Configuration

Y for Colums	Y for Rows
1.43E-1 -> (32,1,1), (32,1,1)	**8.160E-2 ->** (128,1,1), (8,1,1)
1.60E-1 -> (32,1,1), (16,2,1)	**8.193E-2 ->** (128,1,1) , (4,2,1)
1.59E-1 -> (32,1,1), (8,4,1)	**8.412E-2 ->** (128,1,1) , (2,4,1)
1.59E-1 -> (32,1,1), (4,8,1)	**8.184E-2 ->** (128,1,1) , (1,8,1)
1.70E-1 ->(32,1,1), (2,16,1)	-
1.97E-1 ->(32,1,1), (1,32,1)	-

GridX/gridY Colums	GridX/gridY Rows
1.4192E-1 ->(32,1,1), (32,1,1)	**8.1592E-2 ->** (128,1,1), (8,1,1)
1.4226E-1 -> (16,2,1), (32,1,1)	**8.1582E-2 ->** (64,2,1) , (8,1,1)
1.4296E-1 -> (8,4,1), (32,1,1)	**8.1574E-2 ->** (32,4,1) , (8,1,1)
1.4222E-1 -> (4,8,1), (32,1,1)	**8.1566E-2 ->** (16,8,1) , (8,1,1)
1.4262E-1 -> (2,16,1), (32,1,1)	**8.0254E-2 ->** (8,16,1) , (8,1,1)
1.4286E-1 -> (1,32,1), (32,1,1)	**8.0288E-2 ->** (4,32,1) , (8,1,1)
-	**8.1634E-2 ->** (2,64,1) , (8,1,1)
-	**8.1594E-2 ->** (1,128,1), (8,1,1)

Conclusions about the ordering

- A memória de uma GPU está organizada em formato matriz, com linhas e colunas.
- Ordenar por linhas é mais eficiente porque, os dados são lidos e escritos sequencialmente, ou seja, os dados são lidos de uma só vez em vez de acessar a vários endereços de memória separados, enquanto que ordenar por colunas implica os dados serem lidos e escritos em endereços de memória não sequenciais.
- As GPU's são projetas para realizar operações em paralelo, o que faz com que o processo de ordenação possa ser mais rápido e eficiente. Posto isto, podemos concluir que vale a pena usar a GPU no processo de ordenação.

If there were 1 million values...

- Uma maneira de se ordenar 1 milhão de valores pode ser usando o algoritmo de ordenação QuickSort.
- Para implementar esse algoritmo será preciso dividir os dados em blocos e threads.
- Cada thread pode ser responsável por comparar dois elementos e se eles tiverem fora de ordem, troca-os de lugar. Esta mesma comparação/troca pode ser feita de forma paralela.
- Após cada iteração do algoritmo, é necessário sincronizar todas as threads para garantir que todas tenham terminado as suas comparações/trocas antes da próxima iteração
- No final, após a ordenação, desalocar o vetor alocado na memória da GPU