Sorting Sequences of Values

GPU CUDA optimization

Assignment 2 - Jan. 2023

Arquiteturas de Alto Desempenho
Prof. António Rui Borges

universidade
de aveiro

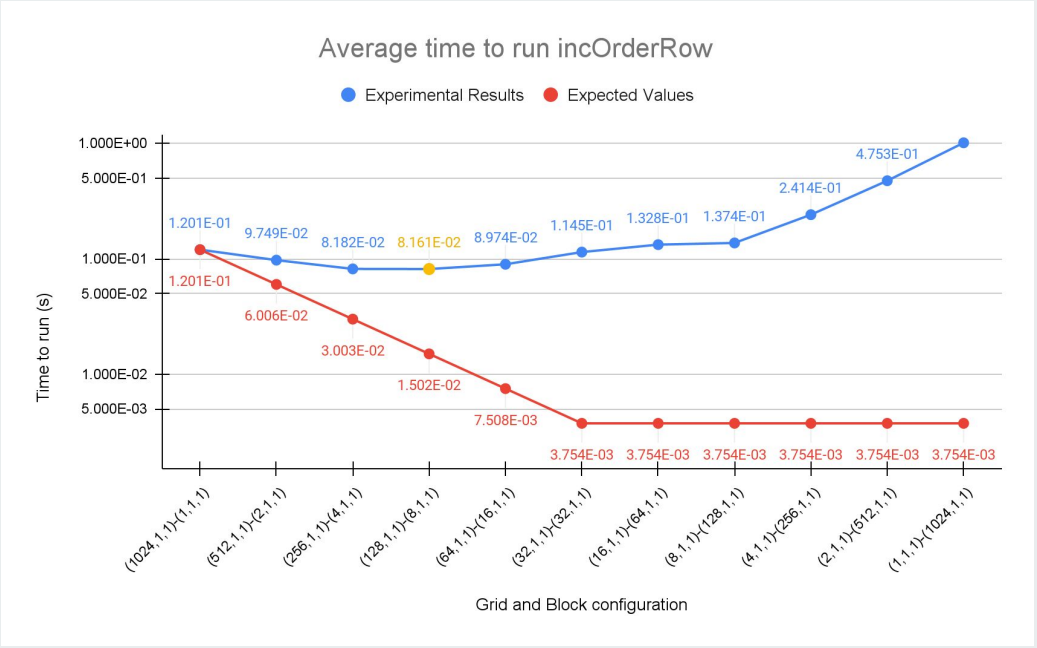
André Clérigo 98485
Pedro Rocha 98256
Turma 1 - Grupo 7

IncOrderRow Optimization

- The optimized launch configuration was <<<(8,16,1), (8,1,1)>>>
- The average time it took for the CUDA kernel to run the program using the best configuration was:

  8.157E-02 seconds
- The average time it took for the CPU kernel to run the program was:

  6.080E-01 seconds



Grid		Block		Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Aver (s)	StDev (s)	stdDev / Aver
X	Y	X	Y								
128	1	8	1	8.164E-02	8.161E-02	8.159E-02	8.165E-02	8.155E-02	8.161E-02	4.02E-05	0.05%
128	1	4	2	8.194E-02	8.194E-02	8.188E-02	8.198E-02	8.196E-02	8.194E-02	3.74E-05	0.05%
128	1	2	4	8.409E-02	8.412E-02	8.409E-02	8.407E-02	8.413E-02	8.410E-02	2.45E-05	0.03%
128	1	1	8	8.188E-02	8.189E-02	8.186E-02	8.194E-02	8.190E-02	8.189E-02	2.97E-05	0.04%
64	2	8	1	8.163E-02	8.156E-02	8.159E-02	8.164E-02	8.166E-02	8.162E-02	4.72E-05	0.06%
32	4	8	1	8.160E-02	8.164E-02	8.157E-02	8.160E-02	8.159E-02	8.160E-02	2.05E-05	0.03%
16	8	8	1	8.161E-02	8.160E-02	8.161E-02	8.165E-02	8.159E-02	8.161E-02	1.91E-05	0.02%
8	16	8	1	8.152E-02	8.158E-02	8.157E-02	8.158E-02	8.159E-02	8.157E-02	2.26E-05	0.03%
4	32	8	1	8.159E-02	8.159E-02	8.154E-02	8.158E-02	8.158E-02	8.158E-02	2.23E-05	0.03%
2	64	8	1	8.161E-02	8.161E-02	8.161E-02	8.163E-02	8.159E-02	8.161E-02	1.51E-05	0.02%
1	128	8	1	8.167E-02	8.167E-02	8.167E-02	8.161E-02	8.159E-02	8.164E-02	4.09E-05	0.05%

IncOrderRow - Conclusions

- Very bad mapping since it doesn't take full advantage of the parallelism, based on the fact that it's using bubble sort. Furthermore, by itself, it's a sorting algorithm with complexity $O(n^2)$.

- The most significant variation was the dimension of the block's X value.

- It's still better to use the GPU, since using it is faster and gets better results than using the CPU. Nevertheless, a classic algorithm with a better performance such as merge sort would perform better, however, the best case scenario would be a parallel sorting algorithm.

```
for (i = 0; i < length - 1; i++)
{ noSwap = true;
  for (j = length - 1; j > i; j--)
    if (data[j] < data[j-1])
      { tmp = data[j];
        data[j] = data[j-1];
        data[j-1] = tmp;
        noSwap = false;
      }
  if (noSwap) break;
}
```

Bubble sort used in incOrderRow

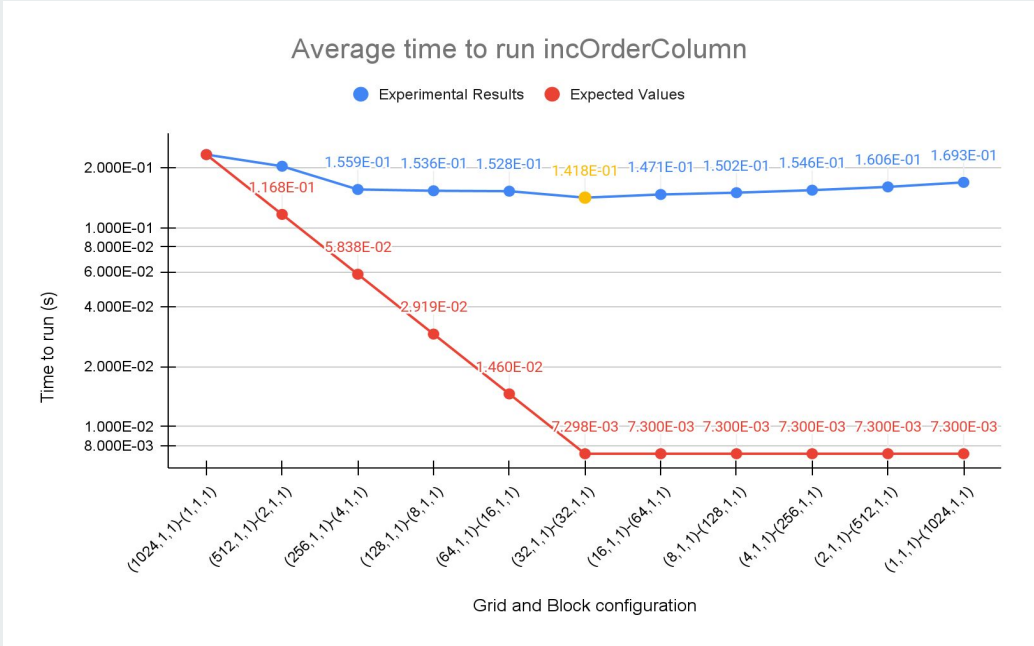
IncOrderColumn Optimization

- The optimized launch configuration was <<<(32,1,1), (32,1,1)>>>
- The average time it took for the CUDA kernel to run the program using the best configuration was:

  1.418E-01 seconds

- The average time it took for the CPU kernel to run the program was:

  8.981E+00 seconds



Grid		Block		Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Aver (s)	StDev (s)	stdDev / Aver
X	Y	X	Y								
32	1	32	1	1.431E-01	1.412E-01	1.410E-01	1.410E-01	1.429E-01	1.418E-01	1.06E-03	0.75%
32	1	16	2	1.595E-01	1.592E-01	1.602E-01	1.596E-01	1.608E-01	1.599E-01	6.39E-04	0.40%
32	1	8	4	1.589E-01	1.589E-01	1.583E-01	1.595E-01	1.584E-01	1.588E-01	4.80E-04	0.30%
32	1	4	8	1.595E-01	1.599E-01	1.579E-01	1.597E-01	1.586E-01	1.591E-01	8.44E-04	0.53%
32	1	2	16	1.727E-01	1.706E-01	1.698E-01	1.699E-01	1.707E-01	1.707E-01	1.17E-03	0.68%
32	1	1	32	1.960E-01	1.969E-01	1.965E-01	1.967E-01	1.969E-01	1.966E-01	3.74E-04	0.19%
16	2	32	1	1.425E-01	1.423E-01	1.431E-01	1.429E-01	1.432E-01	1.428E-01	3.87E-04	0.27%
8	4	32	1	1.431E-01	1.432E-01	1.431E-01	1.431E-01	1.431E-01	1.431E-01	4.47E-05	0.03%
4	8	32	1	1.432E-01	1.431E-01	1.432E-01	1.432E-01	1.432E-01	1.432E-01	4.47E-05	0.03%
2	16	32	1	1.429E-01	1.431E-01	1.431E-01	1.431E-01	1.432E-01	1.431E-01	1.10E-04	0.08%
1	32	32	1	1.431E-01	1.430E-01	1.430E-01	1.431E-01	1.421E-01	1.429E-01	4.28E-04	0.30%

IncOrderColumn - Conclusions and Comparisons

- Again, the most significant variation was the dimension of the block's X value.
- Even though the incOrderColumn program also uses the bubble sort algorithm, it has an even worst mapping than the incOrderRow, since the bubble sort is being used in a non-linear fashion, which is cache-inefficient.
- It accesses the array, that is ordered with the numbers of each row together, in a non-contiguous form (index * N_ARRAYS). Therefore the bubble sort must "jump" from index to index in order to sort the array, creating many cache misses.
- Thus, we should aim for a lower number of blocks, since, on each cache miss, the blocks will try to refresh the cache memory.
- Despite this, it's still better to use this algorithm using the GPU, since the CPU would take much more time to process everything without taking advantage of the parallelization.

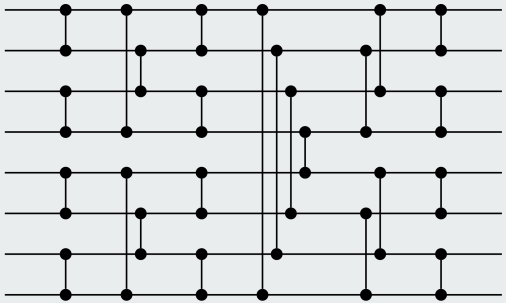
```
for (i = 0; i < length - 1; i++)
{ noSwap = true;
  for (j = length - 1; j > i; j--)
    if (data[j*N_ARRAYS] < data[(j-1)*N_ARRAYS])
      { tmp = data[j*N_ARRAYS];
        data[j*N_ARRAYS] = data[(j-1)*N_ARRAYS];
        data[(j-1)*N_ARRAYS] = tmp;
        noSwap = false;
      }
  if (noSwap) break;
}
```

Bubble sort used in incOrderColumn

Sorting a Single Large Array

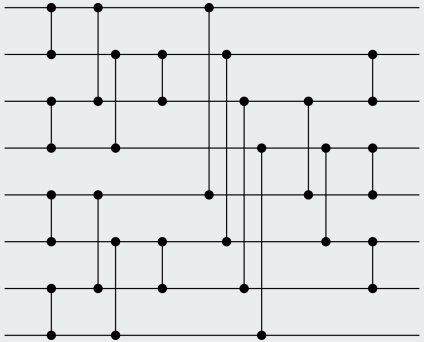
For this problem the length of the single sequence will be equal to N_ARRAYS * ARRAY_LENGTH, which translates to 2^{20} elements. To sort that magnitude of elements we can use oblivious algorithms such as Bitonic Merge Sort or Odd-Even Merge Sort algorithms, these are algorithms that take advantage of GPU parallelism and have $O(\log^2(n))$ complexity for **average**, **best-case** and **worst-case** performance.

Bitonic Merge Sort: To sort an unsorted sequence, we first transform it in a bitonic sequence (sequence that only changes direction once). Starting from adjacent pairs of values of the given unsorted sequence, bitonic sequences are created and then recursively merged into (twice the size) larger bitonic sequences. At the end, a single bitonic sequence is obtained, which is then sorted.



Bitonic sort network

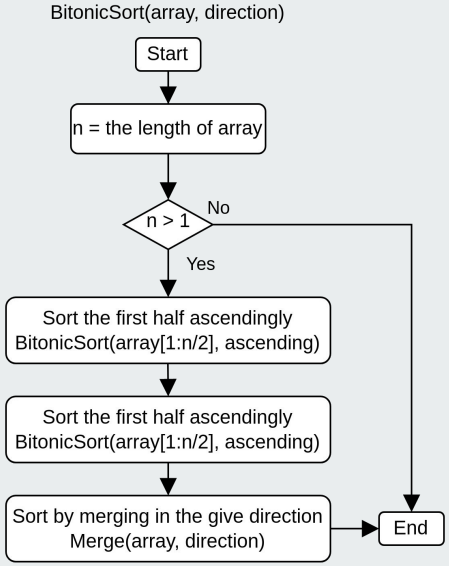
Odd-Even Merge Sort: Odd-even merge sort is a parallel sorting algorithm based on the recursive application of the odd-even merge algorithm that merges sorted sublists bottom up – starting with sublists of size 2 and merging them into bigger sublists – until the final sorted list is obtained.

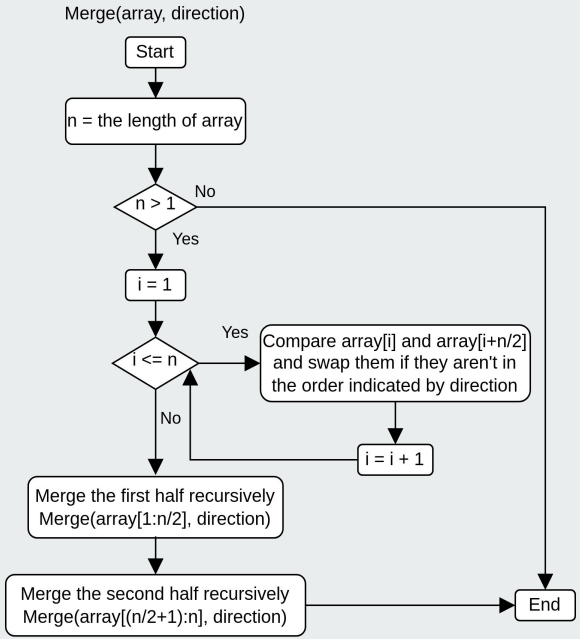


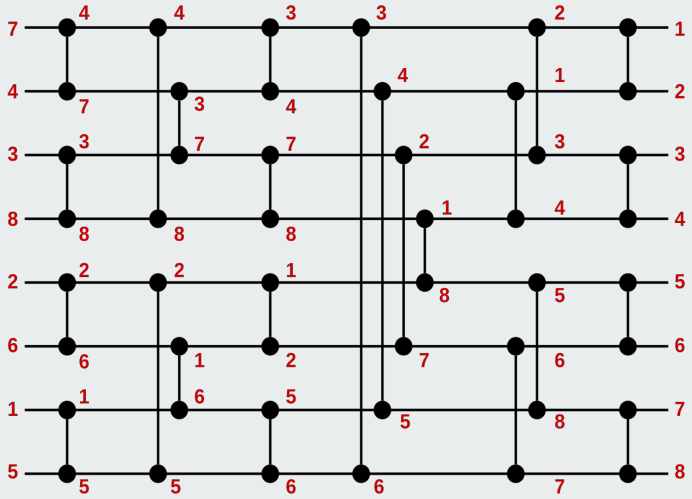
Odd-Even sort network

Sorting a Single Large Array - Sketch

In addition to changing the existing code to support a single array, we need to modify the sorting algorithm. In this case, we will use Bitonic Merge Sort which, based on our research, it is a good algorithm that takes advantage of GPU parallelization because within any set of swaps every swap is completely in parallel. To implement it on a GPU we could assign one thread per input element, and we are actually doing two comparisons, once on their side of the comparison. Something to keep in mind is that we need to synchronized after each comparison, and then begin the next stage.







References

[1]: "Chapter 46. Improved GPU Sorting"

[2]: Wilkinson, B., & Allen, C.M. (2005). Parallel Programming. Pearson/Prentice Hall