Sorting Sequences of Values

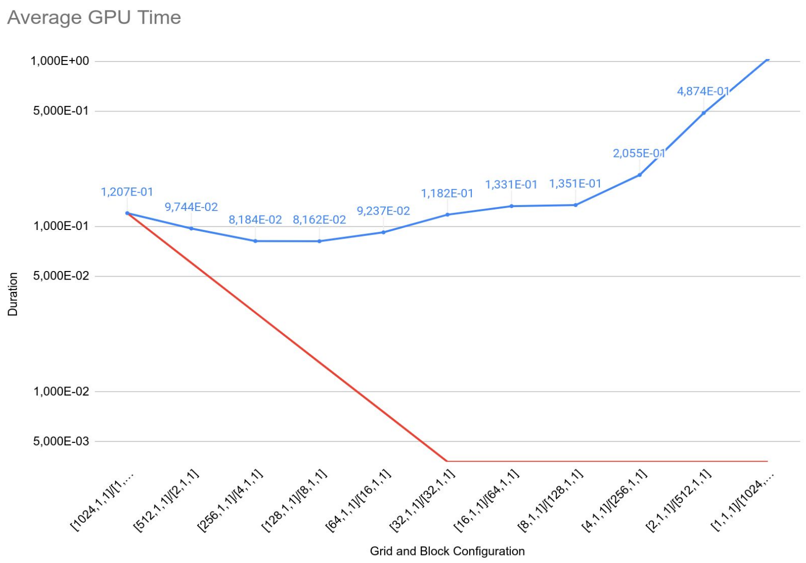
universidade de aveiro
theoria poiesis praxis

Assignment 2 - Arquiteturas de Alto Desempenho
Professor António Rui Borges
jan. 2023

Bruno Lemos	98221
Tiago Marques	98459

Grupo 6 - Turma 1

Rows - Launch Grid Optimization



- Best value was for blocks of 8 threads
- Performance starts to deteriorate for blocks with >= 32 threads

gridDim	blockDim	1 RUN	2 RUN	3 RUN	4 RUN	5 RUN	AVERAGE	STANDARD DEVIATION
[128,1]	[1,8]	8,181E-02	8,185E-02	8,191E-02	8,184E-02	8,189E-02	8,185E-02	4,000E-05
[128,1]	[2,4]	8,412E-02	8,413E-02	8,409E-02	8,414E-02	8,413E-02	8,413E-02	1,924E-05
[128,1]	[4,2]	8,189E-02	8,196E-02	8,193E-02	8,192E-02	8,196E-02	8,193E-02	2,950E-05
[128,1]	[8,1]	8,162E-02	8,166E-02	8,165E-02	8,163E-02	8,165E-02	8,165E-02	1,643E-05

The best time for block organization was obtained with blockDimX=3 and blockDimY=0, corresponding to [8,1] in block organization.

gridDim	blockDim	1 RUN	2 RUN	3 RUN	4 RUN	5 RUN	AVERAGE	STANDARD DEVIATION
[128,1]	[8,1]	8,181E-02	8,185E-02	8,191E-02	8,184E-02	8,189E-02	8,185E-02	4,000E-05
[64,1]	[8,1]	8,159E-02	8,162E-02	8,165E-02	8,161E-02	8,167E-02	8,162E-02	3,194E-05
[32,4]	[8,1]	8,162E-02	8,161E-02	8,163E-02	8,165E-02	8,162E-02	8,162E-02	1,517E-05
[16,8]	[8,1]	8,160E-02	8,163E-02	8,166E-02	8,165E-02	8,159E-02	8,163E-02	3,050E-05
[8,16]	[8,1]	8,157E-02	8,155E-02	8,158E-02	8,158E-02	8,159E-02	8,158E-02	1,517E-05
[4,32]	[8,1]	8,159E-02	8,155E-02	8,162E-02	8,155E-02	8,158E-02	8,158E-02	2,950E-05
[2,64]	[8,1]	8,161E-02	8,164E-02	8,159E-02	8,162E-02	8,162E-02	8,162E-02	1,817E-05
[1,128]	[8,1]	8,160E-02	8,167E-02	8,165E-02	8,159E-02	8,169E-02	8,165E-02	4,359E-05

The best time for grid organization was obtained with gridDimX=3 and gridDimY=4, corresponding to [8,16] in grid organization.

Rows - Conclusions

- Best configuration: <<(8,16),(8,1)>>

- Execution time on GPU is 7,22 times faster than on CPU

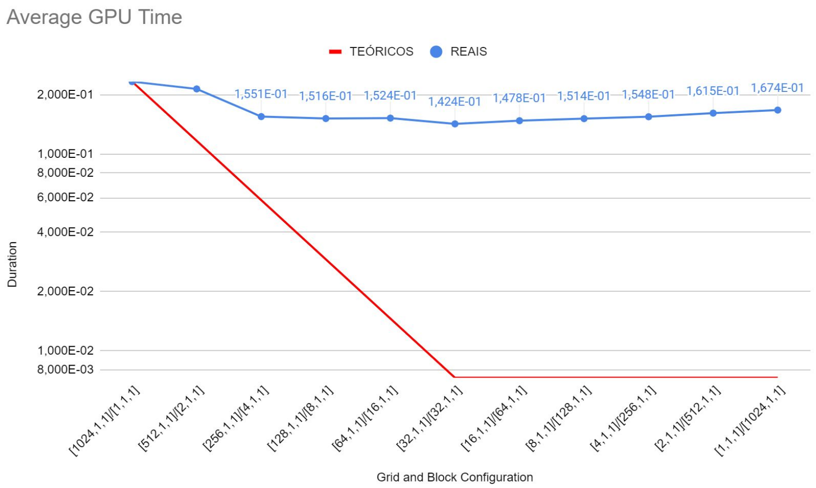
  Average execution time on GPU : 8,158E-02 s
  Average execution time CPU : 5,893E-01 s

- Varying the x and y values of the grid have little impact on performance

- Some cache misses happen
  - The elements of each row are stored consecutively in memory
  - The access to the elements within a row can be considered sequential

- Since we don’t have a lot of cache misses we try to maximize the number of blocks
  - Take advantage of gpu parallelism

Columns - Launch Grid Optimization



- Best value was for blocks of 32 threads
- Performance starts to deteriorate for blocks with < 32 threads.

gridDim	blockDim	1 RUN	2 RUN	3 RUN	4 RUN	5 RUN	AVERAGE	STANDARD DEVIATION
[32,1]	[32,1]	1,286E-01	1,411E-01	1,224E-01	1,430E-01	1,433E-01	1,411E-01	9,546E-03
[32,1]	[16,2]	1,596E-01	1,604E-01	1,599E-01	1,598E-01	1,596E-01	1,598E-01	3,286E-04
[32,1]	[8,4]	1,587E-01	1,583E-01	1,595E-01	1,569E-01	1.597E-01	1,587E-01	1,367E-03
[32,1]	[4,8]	1,639E-01	1,639E-01	1,568E-01	1,601E-01	1,566E-01	1,568E-01	3,577E-03
[32,1]	[2,16]	1,764E-01	1,749E-01	1,703E-01	1,772E-01	1,693E-01	1.740E-01	3,601E-03
[32,1]	[1,32]	1,967E-01	1,970E-01	1,968E-01	1,968E-01	1,972E-01	1,970E-01	1,949E-04

The best time for block organization was obtained with blockDimX=5 and blockDimY=0, corresponding to [32,1] in block organization.

gridDim	blockDim	1 RUN	2 RUN	3 RUN	4 RUN	5 RUN	AVERAGE	STANDARD DEVIATION
[32,1]	[32,1]	1,421E-01	1,431E-01	1,397E-01	1,427E-01	1,411E-01	1,421E-01	1,367E-03
[16,2]	[32,1]	1,425E-01	1,424E-01	1,399E-01	1,423E-01	1,424E-01	1,424E-01	1,120E-03
[8,4]	[32,1]	1,429E-01	1,430E-01	1,418E-01	1,421E-01	1,421E-01	1,421E-01	5,357E-04
[4,8]	[32,1]	1,425E-01	1,422E-01	1,398E-01	1,431E-01	1,420E-01	1,422E-01	1,256E-03
[2,16]	[32,1]	1,419E-01	1,397E-01	1,425E-01	1,424E-01	1,430E-01	1,424E-01	1,290E-03
[1,32]	[32,1]	1,428E-01	1,431E-01	1,419E-01	1,416E-01	1,430E-01	1,428E-01	6,834E-04

The best time for grid organization was obtained with gridDimX=3 and gridDimY=2, corresponding to [8,4] in grid organization.

Column - Conclusions

- Best configuration: <<(8,4),(32,1)>>

- Execution time on GPU is 6,7 times faster than on CPU
  Average execution time on GPU : 1,421E-01 s
  Average execution time CPU : 9,542E+00 s

- Varying the x and y values of the grid have little impact on performance

- A lot of cache misses happen
  - The elements of each column are not stored consecutively in memory
  - The access to these elements cause cache misses most of the time

- Since we have a lot of cache misses we must use less blocks
  - Allows access to the global memory more frequently

Sorting a single big sequence

- Merge sort
  - Divide-and-conquer approach
  - Optimal for large datasets
  - Complexity O(n * log(n))

How it Works ?

1. Divide the array in two halves
2. Recursively sort the two halves of the array
3. Merge the sorted halves for the array together
4. Repeat these steps recursively on the resulting sub-arrays until the entire array is sorted

```
mergeSort(arr, left, right):
    if left > right
        return
    mid = (left+right)/2

    // sort the two halves in parallel
    << grid , block >> mergeSort(arr, left, mid)
    << grid , block >> mergeSort(arr, mid+1, right)

    // merge the two halves
    merge(arr, left, mid, right)

merge(arr, left, mid, right):
    // create temporary arrays
    // copy the data from arr into temporary arrays
    arr = temp
    // merge the temporary arrays back into arr and sort
    arr = merge_values(temp)
    // copy any remaining elements from the left array into arr
    // copy any remaining elements from the right array into arr
    copy(left,right,arr)
```