Batch merge path sort

Antonio Ocello & Jean-Baptiste Gaya

Under the supervision of
Lokman Abbas-Turki

Summary

Summary

1. Mechanism of merge algorithms : theoretical aspect

In this project, we will see algorithms that merge two sorted arrays A and B so that the output is a sorted array of size |A| + |B|. For simplicity, we will only consider the case |A| = |B| :

A

3	6	7	11	11

B

0	2	8	16	22

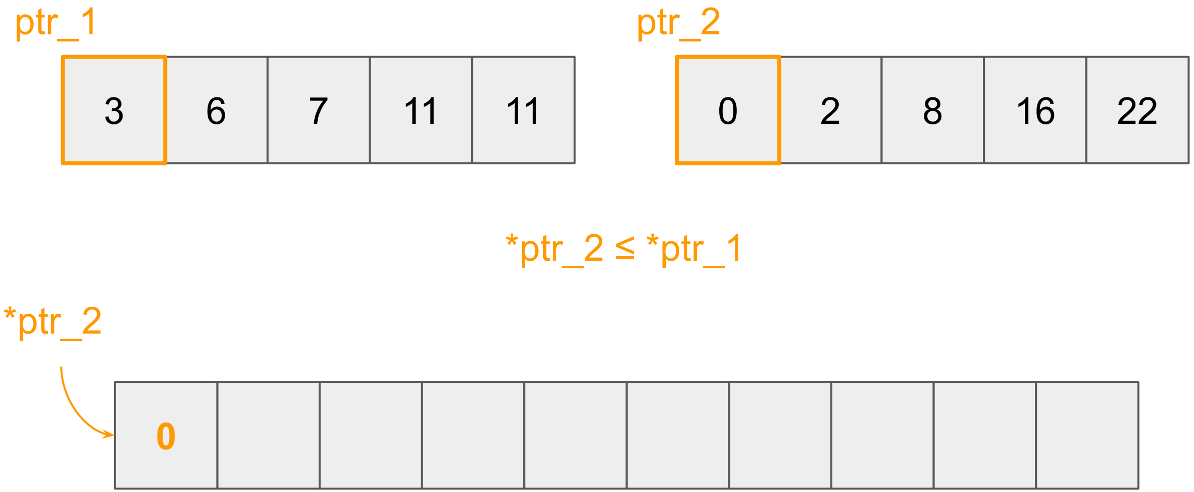
0	2	3	6	7	8	11	11	16	22

We specifically want to compare the efficiency of a **new parallel algorithm** (GPU merge path) to a **classical one** (sequential merge path). After that we will generalize the procedure to a **merge sort case**.

1. Mechanism of merge algorithms : theoretical aspect

1.1 Sequential merge path algorithm

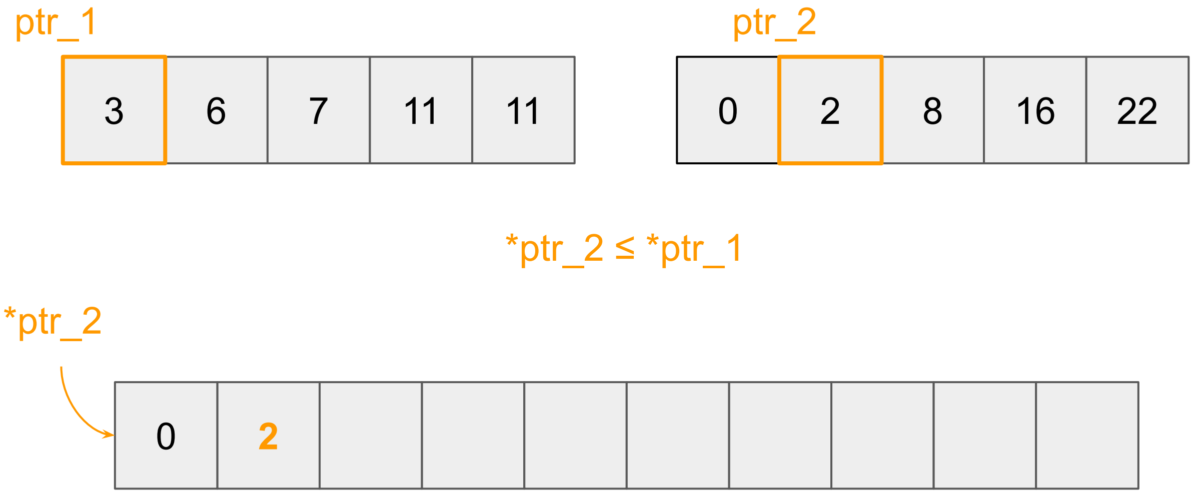
The most common idea is to use two pointers and increment one of them whenever it has the lowest value among both, while copying its value to the output array :



1. Mechanism of merge algorithms : theoretical aspect

1.1 Sequential merge path algorithm

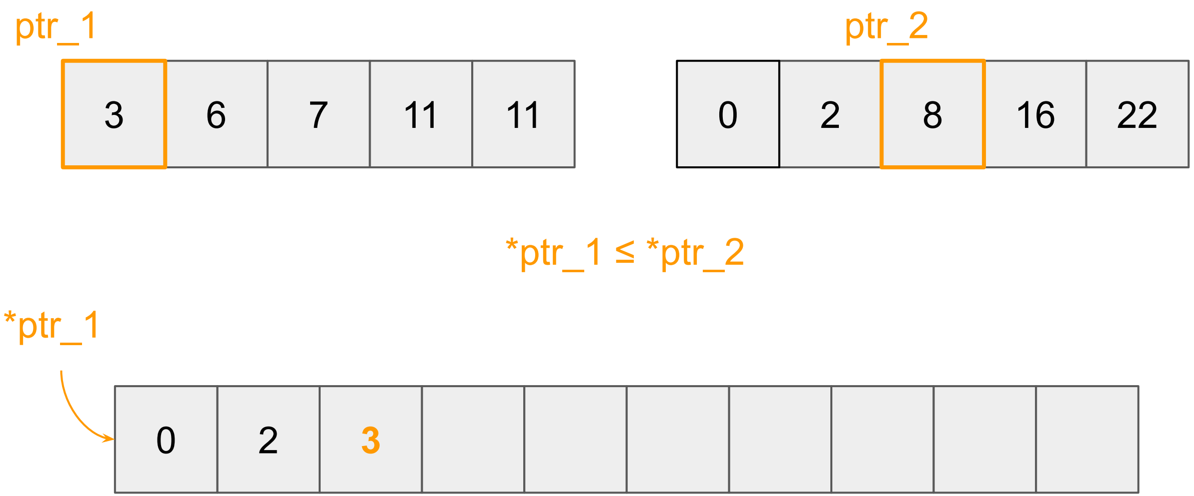
The most common idea is to use two pointers and increment one of them whenever it has the lowest value among both, while copying its value to the output array :



1. Mechanism of merge algorithms : theoretical aspect

1.1 Sequential merge path algorithm

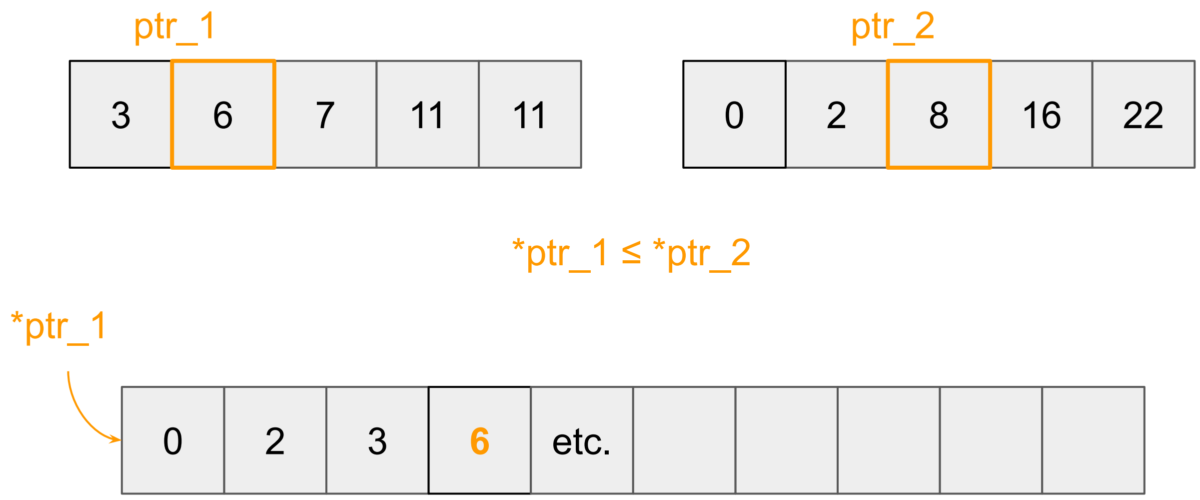
The most common idea is to use two pointers and increment one of them whenever it has the lowest value among both, while copying its value to the output array :



1. Mechanism of merge algorithms : theoretical aspect

1.1 Sequential merge path algorithm

The most common idea is to use two pointers and increment one of them whenever it has the lowest value among both, while copying its value to the output array :



1. Mechanism of merge algorithms : theoretical aspect

1.1 Sequential merge path algorithm

+ It is simple to implement

+ It has always the same linear complexity $O(|A| + |B|)$: no bad surprise

- In its original form, it is impossible to parallelize the algorithm.

- An average linear complexity is not optimal.

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

This recent algorithm from *Green et al. 2012* smartly leverages parallelization by adopting a geometrical point of view. Let’s begin with drawing a boolean matrix C such that

$$\forall \; 1 \leq i, j \leq |A|, \;\; C_{ij} = \mathbf{1}_{A[i] \leq B[j]}$$

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Let’s illustrate it with our previous example :

B

C

A

	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Now, we draw a path separating our matrix C between 0 values and 1 values :

	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Interestingly, this path gives us a way to construct our merged array between A and B :

0									

	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Interestingly, this path gives us a way to construct our merged array between A and B :

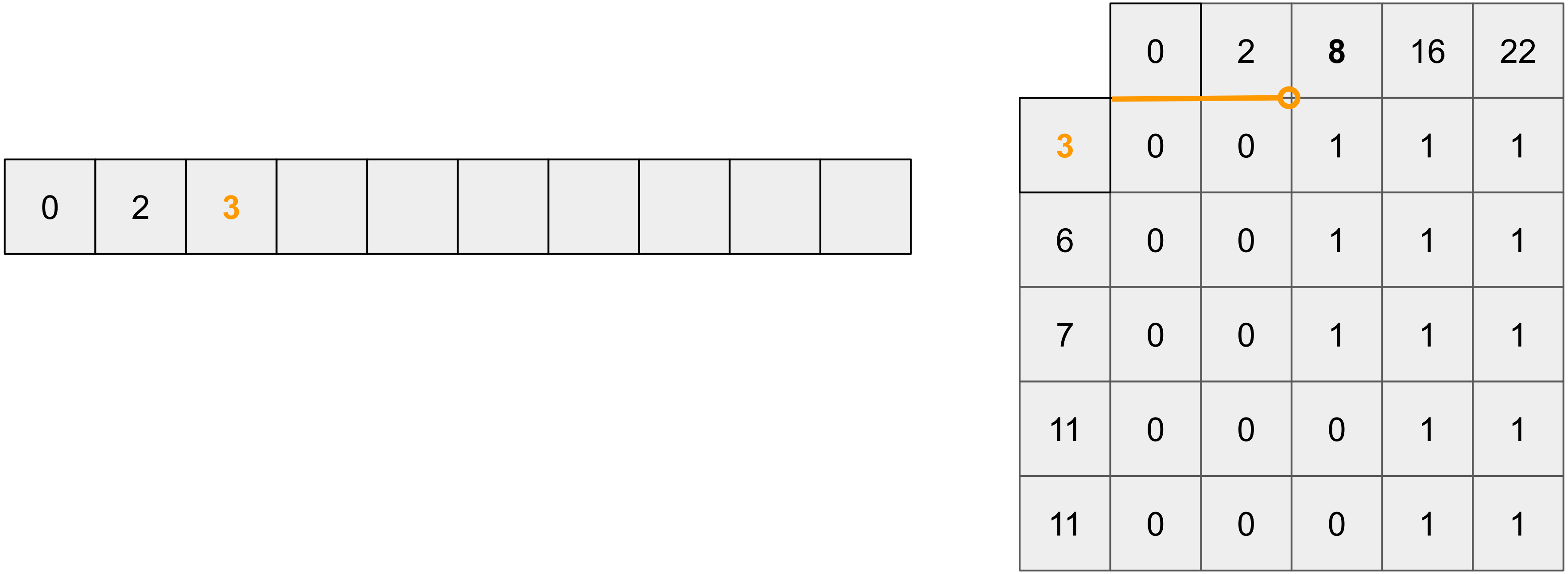
0	2								

	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Interestingly, this path gives us a way to construct our merged array between A and B :



1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Interestingly, this path gives us a way to construct our merged array between A and B :

0	2	3	6						

	0	2	**8**	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Interestingly, this path gives us a way to construct our merged array between A and B :

0	2	3	6	7	etc.				

	0	2	**8**	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

But instead of drawing this path iteratively, we could also try to find the path in a parallel way. For this, we introduce all the diagonals of our matrix C, these will represent the threads (cf next section) in the algorithm :

	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

If we focus on one diagonal, one could imagine an algorithm that would search the intersection between the i-th diagonal and the path, and thus gives us the i-th number to insert in our output array :

	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Thus, we introduce K and P, the extremities of the diagonal. By dichotomy, we evaluate if the middle Q (if the middle is not a point, we take the point above) of the diagonal belongs to the path :

	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

P

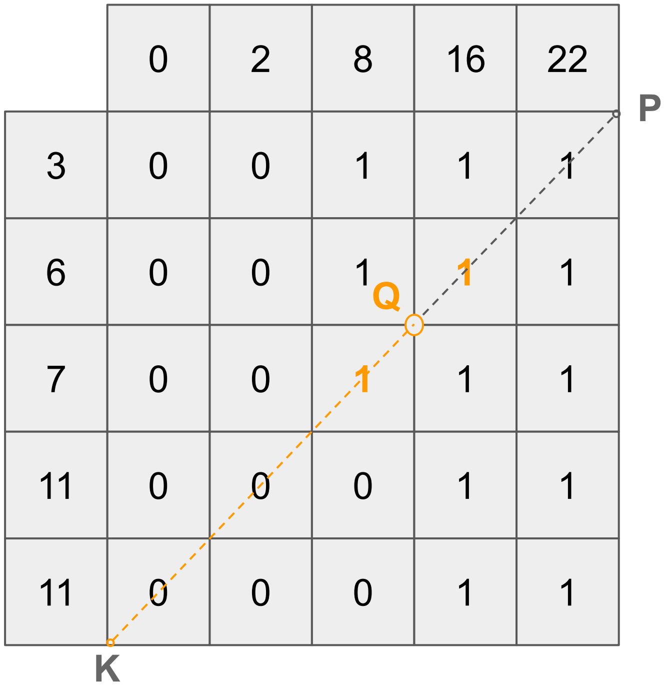
Q

K

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

In our example, the values surrounding Q are equal to one, i.e. the intersection point is on the left side of the diagonal.



1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

We iterate by dichotomy on the left side of the diagonal. Now the values surrounding Q are both equal to 0.

	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

P

Q

K

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

The next step gives us automatically the intersection of the path (and Q is surrounded by 0 and 1 this time).

	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0 Q	**1**	1	1
11	0	**0**	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Then we insert the value in the index of the diagonal (here the 6th) with a simple rule : if the boolean on the down left corner of Q is 0, we take the following value of B, else we take the following value of A :

					Q				
					8				

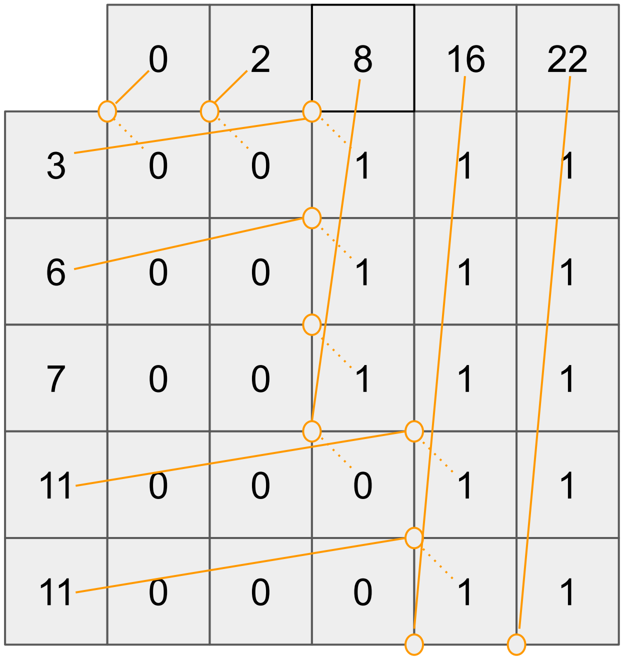
	0	2	8	16	22
3	0	0	1	1	1
6	0	0	1	1	1
7	0	0 Q	1	1	1
11	0	0	0	1	1
11	0	0	0	1	1

1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

Generalizing the procedure gives us the output array :

Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9
0	2	3	6	7	8	11	11	16	22



1. Mechanism of merge algorithms : theoretical aspect

1.2 GPU merge path algorithm

+ It enables parallelization on GPUs.

+ Finding the intersection is cheap : O(log n)

- Without GPU, the complexity is much higher : O(n * log n)

- It is more difficult to implement and requires to think about a strategy to allocate the good number of blocks / thread, especially if the length of the arrays is not a power of 2.

1. Mechanism of merge algorithms : theoretical aspect

1.3 Generalizing to a merge sort algorithm

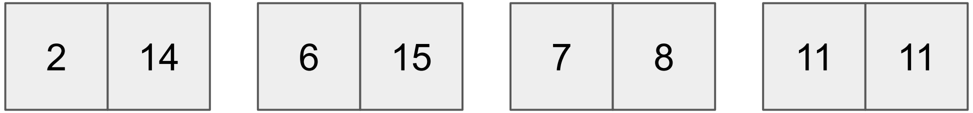
Now suppose we have a unsorted array and we want to leverage merge path to sort it.

14	2	15	6	7	8	11	11

1. Mechanism of merge algorithms : theoretical aspect

1.3 Generalizing to a merge sort algorithm

We only have to initialize merge path by dividing this array in 2-length arrays that we sort with a simple procedure :



1. Mechanism of merge algorithms : theoretical aspect

1.3 Generalizing to a merge sort algorithm

Then, we apply merge path algorithm with a divide and conquer method :

2	6	14	15

7	8	11	11

1. Mechanism of merge algorithms : theoretical aspect

1.3 Generalizing to a merge sort algorithm

Then, we apply merge path algorithm with a divide and conquer method :

2	6	7	8	11	11	14	15

Summary

1. Mechanism of merge algorithms : theoretical aspect

    1.1 Sequential merge path algorithm

    1.2 GPU merge path algorithm

    1.3 Generalizing to a merge sort algorithm

2. The implementation

3. Testing

2. The implementation

Allocation

We allocate an array of size n (which for simplicity we consider to be a power of 2) in the GPU, that we call M_dev.

Initialization (sort)

We divide M_dev into n/2 pairs of values and sort them with a kernel function using n/2 blocks :

```
sort_array<<<n/2,1>>>
```

Merge path

We divide M_dev into nb_arrays arrays of size array_size. Using a while loop, we apply merge path algorithm on all the arrays until nb_arrays ==1 :

```
merge_array<<<nb_arrays,array_size>>>
nb_arrays /= 2
array_size *= 2
```

2. The implementation

Problem : vectors' size larger than 1024
Maximum number of threads per block: 1024.

Solution
We used an indexed stride based on blockDim.x and blockIdx.x in the kernel functions and modified the call of functions in our main program:

```
merge_array<<<nb_arrays,min(array_size,1024)>>>
```

Summary

1. Mechanism of merge algorithms : theoretical aspect

    1.1 Sequential merge path algorithm

    1.2 GPU merge path algorithm

    1.3 Generalizing to a merge sort algorithm

2. The implementation

3. Testing

3. Testing

Three different programs coded
All are following the divide and conquer philosophy.

- Sequential_merge_path.cpp
  classical merge algorithm running on CPU.

- Merge_path.cpp
  merge path algorithm from O. Green et al. running on CPU.

- Merge_sort_path.cu
  parallelized mergesort algorithm on GPU:
  - Sort (initialization) : sortinf first a big array dividing it in arrays of length 2.
  - Merge : Apply merge path from *O Green et al.* on GPU.

3. Testing

Criterion of testing

We decided to compare the wall-clock time of merging randomly generated vectors of integers for the three algorithms. For the GPU algorithm, we also monitored the wall-clock time for sort initialization.

Libraries

We used the libraries random for the CPU versions and cuRAND for the GPU one.

Sanity check

We implemented a sanity check (on CPU for the three versions) to check if the array is really sorted in the end of the program.

3. Testing

Results (merge part)
We ran 100 simulations of the three algorithms, gathering the average wall-clock time of the merge part only. The task consisted in merging 2^20 = 1,048,576 arrays of size 2 :

Algorithm	Wall-clock time (ms)
Merge path (CPU)	50 032
Sequential merge path	1 650
Merge path (GPU)	70

$\approx \times 20$

GPU : GeForce GTX 1060 with Max-Q Design
CPU : Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz

3. Testing

Results (sort part)

We also compared two different ways to call the kernel function sort_array<<< , >>> in the GPU merge path to enlighten the importance of contiguity. The task consisted in sorting 2^20 = 1,048,576 arrays of size 2 :

Kernel call	Wall-clock time (ms)
sort_array<<<n/2,1>>>	2,08
sort_array<<<n/(2*min(n/2,1024)),min(n/2,1024)>>>	0.11

$\approx \times 20$

GPU : GeForce GTX 1060 with Max-Q Design
CPU : Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz

Thank you

Questions ?