

**KU LEUVEN**



FACULTEIT  
INGENIEURSWETENSCHAPPEN

---

# Array flipping

---

## GPU vs CPU

---

Benjamin Rübenkamp

R0793577

MEIsw

Master industrieel ingenieur Elektronica-ICT

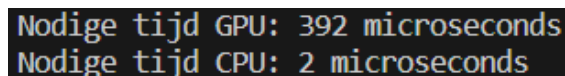
Academiejaar 2023-2024

**\*\* There are no code examples in this text, please look in github repository under session1:  
[https://github.com/Benjamin4-23/Geavanc\\_comp\\_arch/tree/main/session1](https://github.com/Benjamin4-23/Geavanc_comp_arch/tree/main/session1) \*\***

First, an array of length 999 is created, it then resides on memory where only the CPU can access it quickly, but not the GPU. The array is populated with the index of the cell. Once the array is initialized it is also created in GPU memory. (Allocation and copy) Now that the array is available on the GPU, it can perform operations on it in parallel.

A kernel function is written for this purpose. This is given the global keyword because it must be present on both the CPU and GPU. The index of the cell on which the function is executed is calculated as follows: Number of blocks \* size of a block + the thread ID within its block. Once the index of the block is known the function will swap the value in that cell with its mirror image (seen as opposite to the center of the array). Since there is only run over half the array (the first half changes the second half) only half as many threads as cells must be created. After the operation, the new version of the array must be copied back into the CPU memory where the program is running and where it can be printed out if necessary. The time required for the operation is tracked. (Time for copying the data is not counted).

In order to have a comparison, the arrayswap on the CPU is also done once and the time is also tracked.



```
Nodige tijd GPU: 392 microseconds
Nodige tijd CPU: 2 microseconds
```

Figuur 1: Timing GPU vs CPU

The picture above shows that the GPU took longer to perform the operation. This is probably because there is overhead to start all the threads and that each thread has so little to do that the overhead causes a relatively large delay. If each thread had more work, the GPU should be faster by executing in parallel and the startup of the threads should have relatively (to executing operations) less impact.

#### Notes:

- The specification of my GPU only allows me a maximum of 1024 threads in a block.
- (number of blocks \* number of threads per block) must be more than the number of invocations that are needed.