Cooperative Groups: Task 2

Using the result of the previous exercise, this task shows the usage of a statically-tiled partition.

The device-side function `maxFunction`, operating on a group of arbitrary size and returning the groups maximum element, stays exactly the same.

Alterations are to be done in the calling kernel `maxKernel` which should partition the current block into tiles of static size of 16 threads. See the TODOs in the source code.

As usual, compile with `make` and run with `make run`.

Atomic Operations

The code makes use of atomic operations in the final lines of the kernel. Atomic, or *locking*, operations are used when multiple threads access the exact same memory location. Since thread execution is non-deterministic in CUDA it can not be guaranteed that one thread reads/writes a position in memory before another thread. It even can not be guaranteed that a value just read by one thread has not been changed by another thread directly afterwards. Atomic operations lock the memory location for usage by the calling thread and halt other atomic operations until after release of the lock.

Atomic operations come in many flavors but all work in the same manner: The value at a certain address in memory is compared to a given value. Depending on the exact atomic operation, the value at the address is updated or not. Usually, the old value stored at the address is returned. See also CUDA documentation on Atomic Functions.

Examples of atomic functions in CUDA:

- `int atomicAdd(int* address, int val);` Add `val` to the value at `address`; return old value (also available with other data types)
- `int atomicExch(int* address, int val);` Store `val` at `address` location; return old value (also available for `float`)
- `int atomicMin(int* address, int val);` Store the minimum of `val` and the value at `address` at `address` location; return old value
- `int atomicCAS(int* address, int compare, int val);` The value at `address` is compared to `compare`. If true, `val` is stored at `address`; if false, the old value at `address` is stored. The old value at `address` is returned. This is the basic atomic function with which many other atomic operations can be implemented.