Exercise 05 - The Finite Difference Method (3D)

Concepts covered

The following concepts are covered in this exercise

- Iterative improvement of an implementation of a 3D finite difference stencil code.
- Improving performance by optimizing memory usage for GPU code.

Files needed

The following files will be needed for the exercise

- `FDM3D.cu` (supposed to be correct and will not need changes)
- `FDM3D_kernel.cu` (not correct and will need to be changed)

Work steps

You will need to modify the C code supplied in the files. You will implement three versions of the stencil code for the GPU.

Device code

1. In the first implementation (`kernelStencil_v1`) you will perform a direct naive porting of the host code already implemented in `kernelStencil_gold`. In this version, threads read from global memory only. Each thread computed the same point for all the planes of the volume.
2. In the second implementation (`kernelStencil_v2`) shared memory will be used in order to take benefit of the reuse of the neighbor values for the X and Y dimensions. Neighbor values for the Z dimensions will still be loaded from global memory.
3. The third implementation (`kernelStencil_v3`) will use registers to store the values of the Z dimension, in addition to the shared memory used for the X and Y dimensions.

Compilation

Compile the final code using a `Makefile`. In case of compilation errors you will need to debug the code until it compiles successfully.

Execution

Execute your compiled program. If you program executes successfully the output should look like this

```
The Finite Difference Method (3D)
CPU time            : (ms)
PASSED
GPU v1 time compute : (ms)
GPU v1 time memory  : (ms)
GPU v1 time total   : (ms) : speedup x
PASSED
GPU v2 time compute : (ms)
GPU v2 time memory  : (ms)
GPU v2 time total   : (ms) : speedup x
PASSED
GPU v3 time compute : (ms)
GPU v3 time memory  : (ms)
GPU v3 time total   : (ms) : speedup x
PASSED
```