Exercise 04 - Flexible-order finite difference computations (1D)

In this exercise we consider the implementation of flexible-order finite difference (nearest neighbor) approximations of derivatives of a function. These types of operations are quite common in many engineering and scientific applications. The general formula for flexible-order finite difference approximations of the q'th derivative of a function $f(x)$ in one space dimension can be expressed as

$$\frac{\partial^q f}{\partial x^q} \approx \sum_{n=-\alpha}^{\beta} c_n f(x_{i+n})$$

where c_n is finite difference coefficients which can be computed using the supplied C function `fdcoeffF.c` and the function $f(x)$ is evaluated at a discrete grid $x_i = hi$, $i = 0, 1, ..., N-1$, with uniform spacing between grid points of size $h = \frac{1}{N-1}$. α and β are integer values indicating the number of points, respectively, to the left and right of the expansion point x_i. Take $\alpha = \beta$ for all interior points sufficiently far from the boundaries. Near the domain boundaries at x_0 and x_{N-1} the stencils will need to be off-centered.

Concepts covered

The following concepts are covered in this exercise

- How to write GPU kernels.
- How to exploit the memory hierarchy of the device in kernels.
- Performance profiling of incremental improvements to kernel code.
- How to optimize code to maximize utilization of hardware capacity.

Files needed

The following files will be needed for the exercise

- `fdcoeffF.c` (computes finite difference stencils. Supposed to be correct and will not need changes)
- `FlexFDM1D_Gold.c` (Supposed to be correct and will not need changes)
- `FlexFDM1D.cu` (supposed to be correct and will not need changes)
- `FlexFDM1D_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.

1. Familiarize yourself with the host version in `FlexFDM1D_gold.c` for computing the finite difference approximations. Take notes to ensure that your understand the algorithm.
2. Implement a naive kernel `FlexFDM1D_naive()` of the routine coded in `FlexFDM1D_Gold.cu`.

Then, we will try to improve the performance of the naive kernel by the following incremental steps

3. Implement a kernel `FlexFDM1D_v2()` which seek to minimize global memory accesses by utilizing shared memory. This is done by utilizing the threads within a block to cooperate on loading the needed function values to shared memory. Then for each thread do the computation based on function values obtained from the shared memory. Make sure that the code can handle arbitrary number of points N.

4. To further improve overall performance, we will try to minimize data-transfer for the stencils by exploiting the cached constant memory which provides low-latency access to coefficients when all threads in warp access the same constant value.

If time permits, try to evaluate the performance by profiling the code using the CUDA profiler (note that both a text version and a graphical version exist).

5. Try and profile the naive code and compare with the incremental improvements that you have made to the kernels. Evaluate the performance and compare with the theoretical peek memory and compute capacity of the hardware to determine to what extent the resources are utilized.

Execution

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

```
FlexFDM1D
Approximation of first derivative in 1D using the finite difference method.
  ./FlexFDM1D <Nx:default=1000> <alpha:default=3> <THREADS_PR_BLOCK:default=MaxOnDevice>

Device 0: Maximum number of threads per block is ?.
Number of points in x-direction, Nx = ?.
Halfwidth of finite difference stencil, alpha = ?.
Threads per block = ?.

Average timings per kernel invocation:

  CPU time          : ? (ms)
  CPU flops         : ? (Gflops)

  GPU v1 time compute: ? (ms)
  GPU v1 time memory : ? (ms)
  GPU v1 time total  : ? (ms): speedup ?x
  GPU v1 flops       : ? (Gflops)
  PASSED

  GPU v2 time compute: ? (ms)
  GPU v2 time memory : ? (ms)
  GPU v2 time total  : ? (ms): speedup ?x
  GPU v2 flops       : ? (Gflops)
  PASSED

  GPU v3 time compute: ? (ms)
  GPU v3 time memory : ? (ms)
  GPU v3 time total  : ? (ms): speedup ?x
  GPU v3 flops       : ? (Gflops)
  PASSED
```