Exercise 07 - Poisson 2D

In this exercise we will consider the solution of the following *Poisson Problem* in two space dimensions

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y), \quad (x,y) \in [0,1]^2$$

to be solved on a unit square subject to homogenous (Dirichlet) boundary conditions

$$\begin{aligned} u(0,y) &= 0, \quad 0 \leq y \leq 1, & u(1,y) &= 0, \quad 0 \leq y \leq 1 \\ u(x,0) &= 0, \quad 0 \leq x \leq 1, & u(x,1) &= 0, \quad 0 \leq x \leq 1 \end{aligned}$$

at the boundaries of the domain.

We will assume that the exact solution to the problem is

$$u(x,y) = \sin(\pi x)\sin(\pi y), \quad (x,y) \in [0,1]^2$$

To solve the problem, we will need to approximate the equations. For this we use a discretization method referred to as the central difference method. In this method the derivatives are approximated by finite summations of the form

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{h^2}$$

where $h = \frac{1}{N}$ is a uniform grid distance between discrete points on the unit square such that $(x_i, y_j) = (hi, hj)$ for $i, j = 0, 1, ..., N$ with $N+1$ the total number of points in both the $x-$ and $y-$directions. For convenience we choose to define $N = 2^p$.

By replacing the derivative in the Poisson problem with approximation taking at the discrete grid points and rewriting the equation slightly, we obtain the following discrete set of equations

$$U_{i,j} = \tfrac{1}{4}\left(U_{i,j+1} + U_{i,j-1} + U_{i-1,j} + u_{i+1,j} - f_{i,j}h^2\right)$$

For convenience we choose to redefine the scaled term containing a contribution for the right hand side function using the exact solution as

$$b_{i,j} = h^2 f_{i,j} = 4u(x_i, y_j) - u(x_{i-1}, y_j) - u(x_{i-1}, y_{j+1}) - u(x_{i+1}, y_j) - u(x_{i+1}, y_{j+1})$$

which ensure that the exact solution is an exact solution to the discrete set of equations(!), i.e. to

$$U_{i,j} = \tfrac{1}{4}\left(U_{i,j+1} + U_{i,j-1} + U_{i-1,j} + u_{i+1,j} - b_{i,j}\right)$$

To solve this discrete set of equations we can use the *Jacobi Method*

$$U_{i,j}^{[k+1]} = \tfrac{1}{4}\left(U_{i,j+1}^{[k]} + U_{i,j-1}^{[k]} + U_{i-1,j}^{[k]} + u_{i+1,j}^{[k]} - b_{i,j}\right), \quad k = 0, 1, ...$$

starting from some initial guess, e.g. $U_{i,j}^{[0]} = 0$. The convergence rate can be improved such that we need fewer iteration to achieve a given tolerance level by instead re-using latest computed updates to the solution. This is referred to as the *Gauss-Seidel Method* which can be defined as

$$U_{i,j}^{[k+1]} = \tfrac{1}{4}\left(U_{i,j+1}^{[k]} + U_{i,j-1}^{[k+1]} + U_{i-1,j}^{[k+1]} + u_{i+1,j}^{[k]} - b_{i,j}\right), \quad k = 0, 1, ...$$

when lexicographical ordering is used. That is, update in the order of index $i = 0, ..., N$ first and then index $j = 0, ..., N$. However, this method is inherently sequential as it relies on previous updates. Instead, a red-black coloring scheme can be used for improving parallel properties of the algorithm where every second node is updated (red) in a first sweep and then all other nodes in a second sweep.

Concepts covered

The following concepts are covered in this exercise

- How to time parts of code execution.
- How to allocate device memory.
- How to copy data from CPU to GPU and from GPU to CPU.
- How to solve iteratively a partial differential equation.
- How to optimize performance by taking advantage of device memory hierarchy.
- How to combine computations on GPU with those of CPU to deliver final results.
- How changing an algorithm can affect computation time.

Files needed

The following files will be needed for the exercise

- `Poisson2D_gold.c` (supposed to be correct and will not need changes)
- `Poisson2D.cu` (not correct and will need to be changed)
- `Poisson2D_kernel.cu` (not correct and will need to be changed)

Work steps

You will need to modify the C code supplied in the files. Comment/Uncomment the line `#define USE_DEVICE` to switch between the CPU and GPU implementations. You will implement three versions of the iterative method to solve Poisson's problem in parallel on the GPU device.

You will need to modify the C code supplied in the files.

The host code in `Poisson2D.cu` needs to be updated through the steps

1. Allocate arrays for the solution `u_d` and right hand side function `b_d` in device memory.
2. Transfer arrays `u_h` and `b_h` from host to device.
3. Define the number of threads per block and from this a sufficient number of blocks per grid to be used in the invocation of the kernel `Poisson2D_kernel.cu`.
4. Transfer solution vector `u_d` from device to the vector `u_h` in host memory.
5. Free allocated device memory for vectors `u_d` and `b_d`.

The device code in `Poisson2D_kernel.cu` needs to be updated through the steps

6. Modify the kernel for the Jacobi method such that it it naively uses global device memory only.
7. Modify the kernel for the Jacobi method such that threads within a block can collaborate in loading data from global device memory to shared memory for reducing latency and achieve better overall performance.
8. Repeat steps to implement the Red-Black Gauss-Seidel method.

Small performance study

9. Do a small study where you find out how many iteration are needed to compute the solution to achieve an accuracy level of $||e||_2$ <1e-3 using respectively Jacobi's method and the Red-Black Gauss-Seidel Method.
10. Do a small study where you analyze the performance of the kernel compared to the CPU version. For example, split the gpu timing into data transfer time and computation time and compare the relative speedup with and without memory transfer. Examine how the grid configuration into blocks and threads influence the performance.

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

```
Iterative solution of Poisson's problem in two space dimensions
    ./Poisson2D <p:default=4> <maxiter:default=1000> <SMOOTHER:default=0>

Execution ? Method using ?...
Time used: ? s
? iterations completed to achieve a relative defect tolerance of ?
PASSED!
```