Assignment 3

Deadline: Monday 30 November, 10am.

Contents

Part 1 - Improving the 5-point stencil in CUDA

In Assignment 2 we implemented the 5-point stencil in CUDA. In this exercise we want to do three changes to the implementation.

1.) In our implementation we generated an operator $A : \mathbb{R}^{N^2} \to \mathbb{R}^{N^2}$ that for the vector product Ax returned the entries x_k if k was associated with a boundary coordinate, and the 5-point stencil around x_k otherwise. We want to modify this a little bit now. Remember that the operator A discretised the PDE

$$-\Delta u = f$$

for $u \in [0, 1]^2$ with $u = g$ on the boundary for a given boundary function u. We now want to set the boundary values explicitly to zero, that is for each boundary point x_k we always have $x_k = 0$. We can therefore reduce the system to something that only acts on the interior grid points, giving us an operator $\tilde{A} :\to \mathbb{R}^{(N-2)^2} \to \mathbb{R}^{(N-2)^2}$ where the vector $\tilde{x}$ when we apply $\tilde{A}\tilde{x}$ only contains the interior points $u_{i,j}$ for $0 < i, j < N$.

2.) We noticed in Assignment 2 that our implementation was inefficient. We had as many accesses to global memory as we had operations in each thread. We want to fix this by using a thread block to read a block of grid points into shared memory so that within the thread block we only need to access the shared memory for evaluating the 5-point stencil. Be careful on the boundary of your thread block. You will need to read in a slightly larger neighborhood around the grid points associated with the threads in your threadblock so that all neighboring points are in shared memory (unless you are at the border of the grid).

3.) We want to allow a larger class of possible PDE problems, namely so called modified Helmholtz problems of the form

$$-\Delta u + \omega^2 u = f$$

with zero boundary conditions. Hand over an optional `omega` parameter to the kernel so that the kernel evaluates the discretisation of this modified Helmholtz problem. For `omega=0` we just have the original Poisson problem.

Modify the implementation from Assignment 2 by integrating these three changes. Think carefully about the local block sizes and explain your implementation and your choice of thread blocks and memory movements thoroughly. Validate your implementation by comparing to evaluations with the matrix A from the `discretise_poisson` function, where you set the entries of the input vector x associated with boundary values, explicitly to 0. Note that the matrix A from `discretise_poisson` returns a vector $y = Ax$ that contains also the zeros on the boundary. So you need to filter out the boundary values in y for the validation.

Show benchmark results for your new implementation by comparing timing for matrix-vector products from the previous implementation in Assignment 2. For the timings use the %timeit magic command from Jupyter to achieve high accuracy.

Part 2 - Solving modifed Helmholtz problems with CG on the GPU.

(Hint: If you fail to write a GPU implementation in Part 1, you can also write a CPU implementation so that you can still do Part 2. However, you won't receive any Part 1 marks for it.)

Your CUDA function from the first part takes a vector and returns a vector. It is an implementation of a linear operator. In Part 2 we want to hook this up with the CG solver from Scipy so that we can solve actual PDE problems on the GPU. In order to do this Scipy provides the class `scipy.sparse.linalg.LinearOperator`. This class is a wrapper for routines that evaluate matrix-vector products and turns them into objects that can be passed to iterative solvers. Wrap your solution from Part 1 into a LinearOperator so that you can interface it with the built-in Scipy iterative solvers. Once you have done this use the CG function from Scipy to solve the linear system of equations

$$\tilde{A}x = f,$$

where $\tilde{A}$ is represented through your linear operator. For f simply choose the function with all ones. Make plots of the convergence of the relative residual $\|Ax_k - f\|_2 / \|f\|_2$. As solver tolerance you can choose the default value. Plot within the same graph the convergence curves for different values of $\omega > 0$. Try to explain your observations from what you know about the convergence bounds for CG.

Finally, visualize your PDE solution for $\omega = 1$ using `imshow`.

By Timo Betcke