Assignment 2 - GPU Accelerated solution of Poisson problems

Submission Deadline: 8 November 2021, 10am

In this assignment we consider the solution of Poisson problems of the form

$$-\Delta u(x,y) = f(x,y)$$

with $\Delta := u_{xx} + u_{yy}$ for $(x,y) \in \Omega \subset \mathbb{R}^2$ and boundary conditions $u(x,y) = g(x,y)$ on $\Gamma := \partial\Omega$.

For all our experiments the domain Ω is the unit square $\Omega := [0,1]^2$.

To numerically solve this problem we define grid points $x_i := ih$ and $y_j := jh$ with $i,j = 1,\ldots,N$ and $h = 1/(N+1)$. We can now approximate

$$-\Delta u(x_i,y_j) \approx \frac{1}{h^2}(4u(x_i,y_j) - u(x_{i-1},y_j) - u(x_{i+1},y_j) - u(x_i,y_{j-1}) - u(x_i,y_{j+1})).$$

If the neighboring point of (x_i,y_j) is at the boundary we simply use the corresponding value of the boundary data g in the above approximation.

The above Poisson problem now becomes the sytem of N^2 equations given by

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

for $i,j = 1,\ldots,N$.

Task 1 We first need to create a verified reference solution to this problem. Implement a function `discretise(f, g, N)` that takes a Python callable f, a Python callable g and the parameter N and returns a sparse CSR matrix A and the corresponding right-hand side b of the above discretised Poisson problem.

To verify your code we use the method of manufactured solutions. Let $u(x,y)$ be the exact function $u_{exact}(x,y) = e^{(x-0.5)^2+(y-0.5)^2}$. By taking $-\Delta u_{exact}$ you can compute the corresponding right-hand side f so that this function u_{exact} will be the exact solution of the Poisson equation $-\Delta u(x,y) = f(x,y)$ with boundary conditions given by the boundary data of your known u_{exact}.

For growing values N solve the linear system of equations using the `scipy.sparse.linalg.spsolve` command. Plot the maximum relative error of your computed grid values $u(x_i,y_j)$ against the exact solution u_{exact} as N increases. The relative error at a given point is

$$e_{rel} = \frac{|u(x_i,y_j) - u_{exact}(x_i,y_j)|}{|u_{exact}(x_i,y_j)|}$$

For your plot you should use a double logarithmic plot (`loglog` in Matplotlib). As N increases the error should go to zero. What can you conjecture about the rate of convergence?

Task 2 With your verified code we now have something to compare a GPU code against. On the GPU we want to implement a simple iterative scheme to solve the Poisson equation. The idea is to rewrite the above discrete linear system as

$$u(x_i,y_j) = \frac{1}{4}\left(h^2 f(x_i,y_j) + u(x_{i-1},y_j) + u(x_{i+1},y_j) + u(x_i,y_{j-1}) + u(x_i,y_{j+1})\right)$$

You can notice that if f is zero then the left-hand side $u(x_i,y_j)$ is just the average of all the neighbouring grid points. This motivates a simple iterative scheme, namely

$$u^{k+1}(x_i,y_j) = \frac{1}{4}\left(h^2 f(x_i,y_j) + u^k(x_{i-1},y_j) + u^k(x_{i+1},y_j) + u^k(x_i,y_{j-1}) + u^k(x_i,y_{j+1})\right).$$

In other words, the value of u at the iteration $k+1$ is just the average of all the values at iteration k plus the contribution from the right-hand side.

Your task is to implement this iterative scheme in Numba Cuda. A few hints are in order:

- Make sure that when possible you only copy data from the GPU to the host at the end of your computation. To initialize the iteration you can for example take $u = 0$. You do not want to copy data after each iteration step.
- You will need two global buffers, one for the current iteration k and one for the next iteration.
- Your compute kernel will execute one iteration of the scheme and you run multiple iterations by repeatedly calling the kernel from the host.
- To check for convergence you should investigate the relative change of your values from u^k to u^{k+1} and take the maximum relative change as measure how accurate your solution is. Decide how you implement this (in the same kernel or through a separate kernel). Also, decide how often you check for convergence. You may not want to check in each iteration as it is an expensive operation.
- Verify your GPU code by comparing against the exact discrete solution in Task 1. Generate a convergence plot of how the values in your iterative scheme converge against the exact discrete solution. For this use a few selected values of N. How does the convergence change as N increases?
- Try to optimise memory accesses. You will notice that if you consider a grid value $u(i,j)$ it will be read multiple times from the global buffer. Try to optimise memory accesses by preloading a block of values into local shared memory and have a thread block read the data from there. When you do this benchmark against an implementation where each thread just reads from global memory.

Carefully describe your computations and observations. Explain what you are doing and try to be scientifically precise in your observations and conclusions. Carefully designing and interpreting your convergence and benchmark experiments is a significant component of this assignment.

Previous
Assignment 1 - Matrix multiplication in Numba

Next
Assignment 3 - Sparse matrix formats on GPUs

By Timo Betcke