Assignment 3, Deadline 6 December 2019

General Remarks

- The complete assignment must be written as a single Jupyter notebook that contains the code, relevant output, graphcis and all explanations.
- The Notebook must be executable without errors. To make sure that this is the case, before submitting reload the notebook and run it from scratch to make sure it runs fine. Any submitted notebook that does not run correctly will receive an automatic 0 mark.
- While you can (and should) include in your notebooks results from longer runtime, make sure that the parameter choices in your notebook on submission are chosen such that the notebook does not take longer than 2 minutes to run on your computer. We are most likely using much faster machines for marking. But notebooks that have a substantially longer running time will be rejected.
- You must follow in your code PEP8 coding guidelines with the only exception that lines can be longer than 80 characters. But please do not make them much longer than 100 characters so that they are still readable (consider at a soft limit). Failure to adhere to PEP8 coding guidelines can result in substantial point reductions in severe cases.
- Correctness of the codes will count 40% of the mark. You will receive 30% for presentation, which includes sensible graphs and thorough documentation. The remaining 30% will be for efficient coding. However, a submission without documentation will still count as a fail as this is a mandatory component.
- You will be able to submit your Jupyter notebooks from a few days before the deadline onwards to the Moodle course page. A corresponding announcement will be made in time.

In the following OpenCL is required for Question 1, but not for Question 2. Carefully document everything that you are doing and write down your observations. Documentation is crucial for these exercises. When doing convergence plots you should plot the relative residual $\|r - Ax\|_2 / \|b\|_2$ using a semilogy plot in Matplotlib.

Question 1: Investigating splitting schemes We consider the Laplace problem $-\Delta u(x,y) = 0$ for $x, y \in [0,1]^2$ and boundary conditions $u(x,0) = 1$, $u(0,y) = u(1,y) = u(x,1) = 0$. We consider a usual discretistaion of this problem in terms of a 5 point finite difference stencil, leading to the linear system of equations

$$Ax = b,$$

where b encodes the boundary data and A represents the discrete Laplacian operator. We want to investigate solving this problem in different ways.

- Jacobi Iteration: Implement the Jacobi iteration to solve this problem. Remember that a single step of Jacobi iteration for this example at an interior point u_{ij} consists of taking the average of taking the values at all the neighboring nodes.
- Next implement the Gauss-Seidel iteration by noting that Gauss-Seidel works almost the same way as Jacobi. But in your averaging procedure you incorporate previously updated values. Implement two different schemes of evaluating one step of Gauss-Seidel. 1.) Start with the nodes at the bottom of the square and proceed line by line to the top. 2.) Start at the top and proceed line by line to the bottom. Present results for both schemes.

The Jacobi-Sweep (a single update of all elements in the grid) must be implemented in parallel through an OpenCL kernel. The linear system is then solved through repeated calls to this kernel.

For Gauss-Seidel you cannot easily use an OpenCL kernel that parallelises over all elements since it involves a serial update. Therefore, implement a Numba accelerated version of Gauss-Seidel. However, you can implement a relaxed version of Gauss-Seidel by updating a single row in parallel and using updated values from the previous row. This allows paralellisation across a single row. Implement this scheme with OpenCL and compare its convergence with standard Gauss-Seidel.

Question 2: Preconditioned iterative solvers
Consider the Laplace problem

$$-\Delta u = 0$$

on the domain $\Omega := \Omega_1 \cup \Omega_2$ with $\Omega_1 := [-1,0] \times [-1,1]$ and $\Omega_2 := [-1,1] \times [0,1]$ with nonzero boundary conditions $u = f$ on the boundary of Ω This is a classical L-shaped domain.

- Write a routine that for a given grid size parameter h generates a 5 point stencil discretisation of this domain and for a given boundary function f generates the matrix A and right-hand side b of the discrete linear system $Ax = b$ explicitly. The matrix A must use the CSR sparse matrix format.

You must not first create a dense matrix and then convert it to sparse. Scipy has several sparse formats such as COO that are suitable for the generation of sparse CSR matrices.

- Solve the associated linear system directly with CG and visualise the solution. Study the convergence of CG as you increase the number of discretisation points.

- An often effective preconditioner is given by ILU, the Incomplete LU factorisation. The idea that is that an LU decomposition is performed, but too much fill in of nonzero elements through Gaussian elimination is avoided by setting a drop tolerance of the size of values to be discarded. This leads to an pproximation of Gaussian elimination that only needs little more storage than the original matrix. Scipy has in its sparse linear algebra module an spilu routine that implements this. Use the ILU of your system matrix A as an approximate preconditioner and investigate for different values of the drop factors how the convergence of CG changes.