Assignment 2, Deadline 15 November 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.

The following two questions are computational exercises that require you to develop code implementations with PyOpenCL. It is expected that you develop efficient implementations that make use of advanced CPU parallelisation features. If you have problems with OpenCL CPU drivers on your own computers you can use the Microsoft Azure Notebook platform.

Question 1: OpenCL CSR matrix-vector product
Given a sparse matrix in CSR format. Develop a class that derives from Scipy LinearOperator and which is initialised with data, indices, and indptr array that describe the sparse matrix. The class shall provide an efficient OpenCL accelerated matrix-vector product. Make use of efficient parallelisation and SIMD features provided by Intel's AVX2 technology. Make sure that your data movements are efficient. In particular, the sparse matrix data should only be transferred to the device once and not during each matrix/vector product.

Question 2: Solving a Poisson Problem with OpenCL.
We consider the Laplace equation

$$-\nabla^2 u = f$$

on the unit square with zero conditions on the boundary. Based on the finite difference 5 point stencil discretization discussed in the lecture develop a Linear-Operator that takes a vector u of values on an $M \times M$ grid (x_i, y_j) for $x_i = \frac{i}{M+1}$, i $= 1, \ldots, M$ and y_j described in the same way, and returns the approximate evaluation of $-\nabla^2 u$ for this vector. The evaluation of the 5 point stencil should be parallelized using PyOpenCL without storing the matrix explicitly.

The CG iterative solver in Scipy only requires a LinearOperator that implements matvecs. As right-hand side you can use the simple function $f = 1$. For different values of M plot the convergence curve of the residual $r = \|b - Au_j\|_2 / \|b\|_2$ (for these it is best to use a semilogy plot), where u_j is the approximate solution in iterate j. Also, investigate how the number of iterations grows as M increases. Finally, produce nice plots of the solution u on the unit square.

Note, for this question you need not develop AVX2 optimised OpenCL kernels.