M.Sc. in High-Performance Computing
MAP55672 - Case Studies in HPC
Communication-avoiding QR Factorization

Kirk M. Soodhalter (ksoodha@maths.tcd.ie)
School of Mathematics, TCD

Rules

Please create a git repository with your solutions to this assignment, and submit the link to me by email. One folder in your git repository should contain a Matlib Live file or Jupyter notebook file with the part of the assignment you are asked to do in an interpretted language. The second folder should contain all your C code implementing CAQR the second part of the assignment with a makefile the runs everything. by Friday, 21. February, 2025.

Question

In this assignment, you are being asked to build a simple communication-avoiding TSQR-factorisation of a tall, narrow matrix in both Matlab/Python and then in C programming language. Some parameters:

- We restrict our focus to the case that the tall, narrow matrix is distributed onto four processors, like in the example from the lectures;
- For the Matlab part of the assignment, you will not do any parallel programming; you are simply meant to work out the linear algebra of communication-avoiding TSQR-factorisation; you should take advantage of these languages abilities to directly address sub-blocks of a matrix using *sub-indices*. For example, in Matlab, `W(i:j,:)` returns rows `i` to `j` of the matrix `W`. You may use built-in methods to perform local QR-factorisations of the blocks. In Matlab, this would be `[Q,R] = qr(W_bl)`
- For the C programming part of the assignment, you should familiarize yourself with the QR factorisation routines contained in LAPACK. You will use the Householder-based QR LAPACK routine to perform local, on-processor QR factorisations.

1. [30 points] In Matlab or Python, implement a version of the TSQR that divides an input matrix up into four blocks of rows (using row sub-indexing) and computes the QR-factorisation in the way shown in the lectures on communication-avoiding factorisations.

2. [50 points] In C programming language, implement a method called `TSQR()` which executes just the communication-avoiding `QR`-factorisation of a tall-narrow matrix using the technique described in the lecture videos/included technical report. It should use `LAPACK`'s QR factorisation routine for local QR. Rows of the matrix should be evenly distributed onto four compute nodes, as described in the lectures. Set up a (relatively) small test problem to demonstrate that your code works (i.e., that the factorisation is correct).

3. [20 points] Generate a series of random $m \times n$ test matrices for different values of m and n. How does your code scale with respect to each of these dimensions? Please show some scaling plots to answer this question.