CUDA Programming on NVIDIA GPUs
Mike Giles

Practical 7: solving tridiagonal equations

The PCR (parallel cyclic reduction) algorithm for solving tridiagonal equations
is described in lecture 7.

1. Make and run this application.

   It uses a single block with NX threads to perform 10 iterations of the
   following matrix-vector equation:

   $$A\ x^{n+1} = \lambda\ x^n$$

   where A is the tri-diagonal matrix with $2 + \lambda$ on the main diagonal, and
   -1 on the two neighbouring diagonals. (This comes from an implicit-time
   central space discretisation of a parabolic PDE.)

2. Look at the Gold code to satisfy yourself that it is performing the
   calculation described above.

   Then look at the CUDA code and try to convince yourself that it is
   performing the PCR as described in the lecture notes.

3. A deficiency of the current code is that it uses statically allocated shared
   memory and so can't handle blocks bigger than 128 threads. The solution
   to this is to use dynamic shared memory as described in section 7.2.3 in
   the CUDA Programming Guide, and as used in the reduction code in
   Practical 4.

   Modify the code to do this, and check that it gives the correct results.

4. Next, modify the code to perform M independent calculations (each with
   their own starting value for x) using M thread blocks. This is an example
   of the strategy discussed in Lecture 7 of doing multiple 1D problems at the
   same time.

5. Those who have experience of implicit 1D finite difference time-marching
   applications might like to take this further and customise it to something
   of interest to them.

6. Another option is to modify the CUDA code for the case NX=32 to use a
   single warp and move data using shuffles instead of shared memory.