Introduction to CUDA Programming on NVIDIA GPUs
Mike Giles

## Practical 3: finite difference equations

The main objectives in this practical are to learn about thread block optimisation and the optimal way to handle multi-dimensional PDE applications. It also gives an introduction to two approaches to application profiling.

This practical is based on a code which uses Jacobi iteration to solve a finite difference approximation of the 3D Laplace equation. It performs the calculation on both the GPU and the CPU to check that they give the same answers, and also times how long it takes.

What you are to do is as follows:

1. Compile and run the executable `laplace3d`, and see the results produced and the times taken.

2. Read through `laplace3d.cu` and `laplace3d_gold.cpp` (the CPU reference code).

   In particular, note:

   - The grid is cut into pieces of size $16 \times 16$ in the $x-y$ direction, and each thread block uses 256 threads, with each thread processing one element in each 2D plane.
   - In the kernel code, `IOFF, JOFF, KOFF` give the memory offsets in the three coordinate directions.

   The code is relatively short, so try to understand it completely, including the `u1, u2` pointer swapping in the main code, and the construction of the execution grid `(bx,by)`.

   Please ask questions if anything is not clear.

3. Having verified that the GPU code and the Gold code produce the same answers to within machine precision, comment out the running of the Gold code and the comparison of the answers. You should also increase the grid size to $1024^3$ to give a longer execution time, keeping all of the SMs busy for most of the time.

4. Try changing the thread block size to improve the performance – what are the optimal dimensions?

5. Read through the source code for the new version in `laplace3d_new.cu`. Note that in the new version each thread handles a single grid point.

   Again run it first to check that it gives the correct results, then comment out the checks and increase the grid size to $1024^3$.

   Optimise its 3D thread block size, and compare its optimum running time to the original version.

6. Use the NVIDIA command line profiler

   ```
   ncu laplace3d
   ```

   to see what information it gives you.

   Count the number of integer and single precision floating point operations using the following commands:

   ```
   ncu --metrics "smsp__sass_thread_inst_executed_op_fp32_pred_on.sum,
   smsp__sass_thread_inst_executed_op_integer_pred_on.sum" laplace3d
   ```

   ```
   ncu --metrics "smsp__sass_thread_inst_executed_op_fp32_pred_on.sum,
   smsp__sass_thread_inst_executed_op_integer_pred_on.sum" laplace3d_new
   ```

   (Note: you may have trouble doing a cut-and-paste of these instructions from this PDF because the underscores may not get copied across.)

   The number of integer operations is surprisingly high – I think this is due to index arithmetic for the array references.

7. Estimate how much data is moved from the device memory into the GPU, and from the GPU back to the device memory, in each iteration.

   Given the execution time per iteration, what device memory bandwidth does this imply?

   This old NVIDIA blog article says the effective bandwidth is the sum of the read and write bytes, divided by the execution time; use this in working out your effective bandwidth.

   Is this a good fraction of the peak bandwidth capability of the hardware?