Introduction to CUDA Programming on NVIDIA GPUs
Mike Giles

Practical 4: reduction operation

The main objectives in this practical are to learn about:

- how to use dynamically-sized `shared` memory
- the importance of thread synchronisation
- how to implement global reduction, a key requirement for many applications
- how to use shuffle instructions

What you are to do is as follows:

1. Read through the `reduction.cu` source file and note the following:
   - The main code computes the results using both the CPU and the GPU. The CPU code is very simple, whereas the GPU code is much more complex.
   - Try to understand the `reduction` kernel completely.
   - The kernel uses dynamically allocated shared memory; the size is a third argument in the `<<< >>>` brackets.

2. Compile and run the executable `reduction`, and check that it gets the correct result.

3. The code currently assumes the number of threads is a power of 2.

   Extend it to handle the general case by finding the largest power of 2 less than `blockSize`, and adding the elements beyond that point to the corresponding first set of elements of that size. Test it with 192 threads.

   Rounding `n/2` up to the nearest power of 2 (or equivalently rounding `n` up to the nearest power of 2 and then dividing by 2) can be accomplished with the following code:

   ```
   int m;
   for (m=1; m<n; m=2*m) {}
   m = m/2;
   ```

(On the course webpage I have provided a little program which demonstrates other ways of doing this rounding up which are more efficient but also more obscure.)

4. The code currently performs the reduction operation for a single thread block. Modify the code to perform reduction using multiple blocks with each block working with a different section of the input array.

   As explained in Lecture 4, there are two ways in which the partial sums from each block can be summed:

   - each block puts its partial sum into a different element of the output array, and then these are transferred to the host and summed there;
   - an atomic addition or lock is used to safely increment a single global sum.

   Try at least one of these.

5. Modify the block-level reduction to use shuffle instructions as described in Lecture 4.

6. If there is time, modify the `laplace3d` example from Practical 3 to compute the root-mean-square change at each timestep. This will require a global reduction to sum the squared changes.