LSU EE 7700-2 Homework 2 Due: 11 March 2011

Preliminary

For this assignment check out the code using: `cd ~`
`svn co https://svn.ece.lsu.edu/svn/gp/hw/hw2`

Problem 1: Compile and run the stencil-2d code on one of the workstations in the undergraduate laboratory.

(*a*) Determine the ratio of FP computation to data needed by the stencil 2D code. The number should only include FP operations used in computing the value written to `b[idx]`, ignore FP or other calculations used for computing things such as the number of blocks per row. *Hint: This number has been given in class many times.*

(*b*) Find the computation to data achievable by the GPU, this number is printed by the stencil-2D code.

Problem 2: Repeat the block count experiments for stencil-shared. That is, run configurations with a fixed number of threads per multiprocessor but with varying block sizes. The configurations should have the minimum number of threads needed to achieve peak performance. Based on this determine whether the number of blocks matters for a fixed number of threads. Do this both for `stencil_iter` and `stencil_shared`.

Problem 3: The code in `stencil_shared_blocked_2` and `stencil_shared_blocked` are identical. For this problem improve `stencil_shared_blocked_2` by having a single thread operate on more than one element per iteration. Consider the loop below:

```
while ( idx < idx_stop )
  {
    int idx_next = idx + array_row_stride;
    s[sid] = a[idx_next];
    __syncthreads();
    if ( !load_only )
        b[idx] = v0 * s[sidx]
          + v1 * ( s[sidx-1] + s[sidx+1] + s[siu] + s[sid] )
          + v2 * ( s[siu-1] + s[siu+1] + s[sid-1] + s[sid+1] );
    __syncthreads();
    int sid_new = siu; siu = sidx; sidx = sid; sid = sid_new;
    idx = idx_next;
  }
```

Each iteration operates on one element, and also has the overhead of two synchronizations. Successive iterations operate on successive rows. Modify the code so that an iteration above loads several elements to shared memory, call the number R. All of these elements should be in the same row, and as a result a block of size B will load RB elements. It will then do the computation on R elements and write the results. The code should only have to synchronize once for each group of R elements.

The value of R will be limited by shared memory size.

(*a*) Modify the code as described above in which R is chosen to be the largest possible value (based on shared memory available and any other factors).

(*b*) Modify the code so that it runs for a constant R. That is somewhere there might be code such as:

```
if ( homework_R == 4 )
{
  // Code written for R == 4
}
```

In this code try to eliminate as much instruction overhead as possible, for example by hand unrolling the loop or trying the `#pragam unroll 4` compiler directive (see the CUDA programmer's guide).