Chun Che Wang and Silao Xu (ccwang and silao-xu)
CS 157 — Spring 2013 **Homework HW 9** April 29, 2013

Problem 5

1. (a) 200 ns $\times$ 32
   (b) 200 ns (assume it is aligned to 128-byte boundaries)
2. (a) 2^8 bytes
   (b) 32 KB
3. (a) 32 ns
   (b) 1 ns
4. ```
   Input :
   g_I = 32x32 array
   g_O = 32x32 array
   arr_dim = 32

   Transpose(array g_I, array g_O, arr_dim)
   {
       register tid = ThreadNumber();
       register x = tid % arr_dim;
       register y = floor( tid / arr_dim ) ;

       share_data[ x + arr_dim*y ] = g_I [ x + arr_dim*y ];

       // wait for all the data to be loaded in share memory
       syncthreads();

       // Do the matrix transpose in share memory
       g_O[ x + arr_dim*y ] = share_data[ y + arr_dim*x ];
   }
   ```
5. ```
   Input :
   g_I = 32x32 array
   g_O = 32x32 array
   arr_dim = 32

   Transpose(array g_I, array g_O, arr_dim)
   {
       register tid = ThreadNumber();
       register x = tid % arr_dim;
       register y = floor( tid / arr_dim ) ;

       // pad the share memory by one more column to avoid bank conflicts
       share_data[ x + (arr_dim+1)*y ] = g_I [ x + arr_dim*y ];
   ```

```
    // wait for all the data to be loaded in share memory
    syncthreads();

    // Do the matrix transpose in share memory
    g_O[ x + arr_dim*y ] = share_data[ y + (arr_dim+1)*x ];
}
```

6. Execution configuration:

```
block size = 1024
number of blocks = 128*128

Input :
g_I = 4096x4096 array
g_O  = 4096x4096 array
arr_dim = 4096

Transpose(array g_I, array g_O, arr_dim)
{
    register block_dim = 32;
    register banks = 32;

    // x,y coodinate for each block
    // ex: number 1 block --> (0,0), number 2 block --> (0,1),.. etc
    register x_block = blocknumber() % 128;
    register y_block = floor(blocknumber() / 128);

    // offset for g_I
    register offset = (x_block*block_dim) + (y_block*block_dim)*arr_dim ;

    // x,y coordinate for threads in the block
    register x_thread = ThreadNumber() % block_dim;
    register y_thread = floor( ThreadNumber() / block_dim);

    // pad the share memory by one more column to avoid bank conflicts
    share_data[ x_thread + (banks+1)*y_thread ] =
                              g_I [ offset + x_thread + arr_dim*y_thread ];

    // wait for all the data to be loaded in share memory
    syncthreads();

    // offset for g_O
    register t_offset = (x_block*block_dim)*arr_dim + (y_block*block_dim) ;

    // Do the matrix transpose in share memory
    g_O[ t_offset + x_thread + arr_dim*y_thread ] =
                              share_data[ y_thread + (banks+1)*x_thread ]; }
```

7. In Part (4), we make use of the share memory (32×32) to do the matrix transpose (since accessing the share memory is much faster than accessing the global memory, 200ns vs 1ns), instead of doing something like $g\_o[j * 32 + i] = g\_I[i * 32 + j]$ which g_o may stride in memory. This strategy speed up the execution by reducing the frequency of accessing the global memory.

   In Part (5), for a share memory (32×32), all the elements in a column of data map to the same shared memory bank, this would result in 32-way bank conflict. We solve this by padding one more column to the share memory to avoid the bank conflicts, namely we stote the data from global memory to the share memory that is allocated as 32×33 bytes.(the last column is always useless).

   In Part (6), we allocate 128*128 blocks of threads, and each block size is 1024. We map each block to an (x,y) coordinate according to their block number, for instance: 1_{st} block map to (0,0), 2_{nd} block map to (0,1), 128_{th} block map to (1,0),..etc. Therefore, after transposing the (x,y) block using the part (5), we write the data in its share memory back to the block(y,x) in g_o. By running hundred of thousands threads parallely, we can hide the memory latency to speed up the matrix transpose in really large dimension.

8. Simultaneously, there are 1024 (8 processors with 128 cores) threads running parallely on the code. Further, according to part 4 and 5, the warp of 32 threads must collectively touch one 128 byte block of global memory, and also they much touch each of the 32 banks of shared memory exactly once. Therefore, each thread is accessing the global memory (200ns) only twice and twice with the share memory (1ns) as well. The total time for these threads to finish the task is about

   $$200 \times 2 + 1 \times 2 = 402\text{ns}$$

   Our input array have 2^{24} entries and we set 1 thread per entry of the matrix, the time for all the entry to get transposed is

   $$2^{24}/2^{10} \times 402 \times 10^{-9} = 0.006585386$$

   The running time is about 10 times slower than the optimal one (0.00064 sec), however this is way better than transpose the matrix naively(stride in memory access), which we spend lots of time accessing global memory). In CUDA programming we make use of hundred of thousands threads running parallely to hide the memory latency.

Problem 2

we can take advantage of the power of eigenvectors and eigenvalues to cluster/group the data based on their relationships of each other(in this case, the relationship is represent in matrix fr. We can further change the matrix fr to other representation. For instance, each (i, j) in the matrix can now represents the similarity between i, j. Then, the variety measurement of similarity lead to more interesting applications. There are lots of interesting algorithm similar to this problem such as K-means clustering and spectral neighborhood algorithm.

Problem 3

By random sampling the 16×16 image in a image and compute it with singular decomposition, we can get a series of components in order of its importance (namely, first principal component accounts for as much of the variability in the data). In addition, we accidentally notice these components are in fact Fourier transform basic functions (Wow!). These components can be used for image compression and even with face recognition (Eigenface or Eigenspce). There is a really interesting articles using Eigenspaces to analyze the faces in Miss Korea 2013 Contestants: http://jbhuang0604.blogspot.com/2013/04/miss-korea-2013-contestants-face.html