CSCI-GA.3033-004
Graphics Processing Units (GPUs): Architecture and Programming
Homework Assignment 2 (total: 30 points)

1. Assume the following piece of code is running on G80 (We saw many examples about G80 and its properties in class.):

```
 1  #define VECTOR_N 1024
 2  #define ELEMENT_N 256
 3  const int DATA_N          = VECTOR_N * ELEMENT_N;
 4  const int DATA_SZ         = DATA_N * sizeof(float);
 5  const int RESULT_SZ       = VECTOR_N * sizeof(float);
    ...
 6  float *d_A, *d_B, *d_C;
    ...
 7  cudaMalloc((void **)&d_A, DATA_SZ);
 8  cudaMalloc((void **)&d_B, DATA_SZ);
 9  cudaMalloc((void **)&d_C, RESULT_SZ);
    ...
10  scalarProd<<<VECTOR_N, ELEMENT_N>>>(d_C, d_A, d_B, ELEMENT_N);
11
12  __global__ void
13  scalarProd(float *d_C, float *d_A, float *d_B, int ElementN)
14  {
15      __shared__ float accumResult[ELEMENT_N];
16      //Current vectors bases
17      float *A = d_A + ElementN * blockIdx.x;
18      float *B = d_B + ElementN * blockIdx.x;
19      int tx = threadIdx.x;
20
21      accumResult[tx] = A[tx] * B[tx];
22
23      for(int stride = ElementN /2; stride > 0; stride >>= 1)
24      {
25          __syncthreads();
26          if(tx < stride)
27          accumResult[tx] += accumResult[stride + tx];
28      }
30      d_C[blockIdx.x] = accumResult[0];
31  }
```

a.[1] How many threads are there in total?

b.[1] How many threads are there in a warp?

c. [1] How many threads are there in a block?

d. [1] How many global memory loads and stores are done for each thread?

e. [1] How many accesses to shared memory are done for each block?

f. [2] How many iterations of the for loop (Line 23) will have branch divergence? Show your derivation.

g. [3] Identify an opportunity to significantly reduce the bandwidth requirement on the global memory. How would you achieve this? How many accesses can you eliminate?

2. [3] What factors can make two threads corresponding to two different warps but of the same block take different amount of time to finish? To get full credit, write at least 3 factors.

3. [2] What is the difference between shared memory and L1 cache?

4. [2] Can memory be coalesced for threads in a warp yet not-coalesced for threads in a different warp of the same block?

5.[3] Suppose you want to write a kernel that operates on an image of size 400x900 pixels. You want to assign one thread to each pixel. Your thread blocks are square, in geometry, and try to use the maximum number of threads per block possible on the device. The maximum number of threads per block is 1024. How would you select the grid dimensions and block dimensions of your kernel?

6. Suppose an NVIDIA GPU has 8 SMs. Each SM has 32 SPs, but a single warp is only 16 threads. The GPU is to be used to add two arrays element-wise. Assume that the number of array elements is 2^{24}. Let t denote the amount of time it takes one thread (yes, just one) to perform the entire calculation on the GPU. The kernel code is shown below (num_threads is the total number of threads in the whole GPU):

```
__device__ void prob(int array_size) {
        int tid = threadIdx.x + blockIdx.x * blockDim.x;
        for ( int i=tid; i<array_size; i += num_threads )
                result[i] = a[i] + b[i];
        }
```

(*a*) [1] What is the amount of time it takes if we use one block of 16 threads?

(b) [1] What is the amount of time it takes if we use two blocks of 8 threads each?

(c) [2] Justify why the above two answers are similar/different.

(d) [1] Assume that 256 threads are enough to keep all SPs in the SM busy all the time. What is the amount of time it would take to perform the computation for one block of 1024 threads? Justify.

(e) [1] Repeat question (d) above but with two blocks of 512 threads each.

7. [2] The line of code below checks for a special case to avoid calling an expensive square root. Describe a situation in which it makes sense for CUDA to do that, and a different situation when it makes no sense (meaning it would be faster to do the square root all the time). Assume that 50% of the time d is equal to 1.

```
if ( d == 1 ) s = 1; else s = sqrt(d);
```

8. [2] What is wrong with that piece of kernel code? How to deal with it if we need to syncthreads both in the if-body and else-body (i.e. ho to change that code yet preserve the semantic)?

```
if {
   ….
__syncthreads() ;
}
else {
      ….
__syncthreads() ;
}
```