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

1.

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

VECTOR_N blocks * ELEMENT_N threads / block = 256*1024 threads in total

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

Current GPUs assign 32 threads to a warp

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

ELEMENT_N = 256 threads (second configuration argument of the kernel launch on line 10)

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

Every thread performs two global memory loads on line 21 (one each from A and B). Every thread also writes one value on line 30. Therefore, each thread performs 3 global memory accesses in total.

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

On line 21, 256 writes (one for each thread) to shared memory occur. When a thread executes line 27, it performs two shared memory reads, adds the two values together, and then performs a shared memory write to store the result.
Given the loop structure, on the first iteration, 128 threads will execute the statement, then 64 on the following statement, and so on.
Finally, each thread reads element 0 on line 30, accounting for another 256 accesses. Therefore, the total number of accesses is
256 + (128 + 64 + 32 + 16 + 8 + 4 + 2 + 1)*3 + 256 = 256 + 255*3 + 256 = 1277 accesses.

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

As described in the solution to (e) above, on the first iteration, 128 threads will execute statement 27, while 128 will not. On the second iteration, 64 threads will execute statement 27, while 192 will not, and so on. Furthermore, the threads executing statement 27 on each iteration are threads with indexes of 0 to stride-1: a contiguous set. Divergence will only occur in iterations where stride is not a multiple of the warp size, or 32. We can see that there are five such iterations, specifically the last five iterations of the loop.

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?

Without any conditional statement guarding it, every thread will execute line 30. This means that every thread in the block will be redundantly writing the same final result to the same final location. These writes get serialized in the memory system, and take significantly more bandwidth than is necessary.
This can be avoided by selecting one thread only to perform the copy from shared to global memory.
An example code addition could look like:

```
if (tx == 0)
  d_C[blockIdx.x] = accumResult[0];
```

2. There are many factors, including:

- cache misses by one thread and hits by another
- non-uniform memory access
- thread divergence in one warp but not in another
- The warp where the first thread corresponds take a path in if-else that has high latency instructions while the other warp takes a different path. That is, warp 1 threads follow the if-path while warp 2 threads follow the else-path and one of the paths has longer latency operations.

3. Shared memory is managed by the program whereas the L1 cache is managed by the hardware.

4. Yes, if one warp takes a path in its execution that has memory access that cannot be coalesced while the other warp takes a different path (if-else scenario).

5. The block dimensions will be 32x32. The grid dimensions will be ceil(400/32) x ceil(900/32) = 13x29.

6.

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

***Execution time is* t/16.**

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

***Execution time is* t/16.**

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

The total number of threads is 16 in both cases. The one-block configuration would run on one SM and would be able to use 16 cores, the two-block configuration would run on two SMs and would be able to make use of 8 SPs each, for a total of 16.
Since there are enough SPs for all threads and none of the threads will have to wait for another thread (i.e. threads are independent) , the rate of computation will be the same as the work is divided evenly among threads, the execution time in both cases will be t/16.

(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.

There would be no difference in performance in configurations from 256 to 1024 threads per block. Then the time is t/256.

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

Since there are two blocks two SMs will be used and so the time will be based on 512 threads, so the execution time is t/512.

7. [2] It makes sense if there is no branch divergence, meaning that for all warps the branch is either always taken or always not taken.
If there is branch divergence then the code will run more slowly than code always performing the square root since within a warp the execution time will be the sum of both paths through the if.

8. [2] May result in deadlock if some threads in the warp go to the if-part and others in the else-part.
Move _syncthreads() outside the if-else part.