1. The block dimensions will be 32x32 (which makes 1024 threads). The grid dimensions will be \( \text{ceil}(400/32) \times \text{ceil}(900/32) = 13 \times 29 \).

2. The scenario cannot happen! Those 8 threads will be in the same warp. Threads of the same warp execute their instructions in lock-step fashion, hence make progress together. That is, the warp won’t move to the following instruction unless all threads in the warp are done with the previous instruction.

3. 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
   - A warp whose threads 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.

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

5. a. How many threads are there in total?

   \( \text{VECTOR}_N \text{ blocks} \times \text{ELEMENT}_N \text{ threads / block} = 256 \times 1024 \text{ threads in total} \)

b. How many threads are there in a warp?

   Current GPUs assign 32 threads to a warp

c. How many threads are there in a block?

   \( \text{ELEMENT}_N = 256 \text{ threads} \) (second configuration argument of the kernel launch on line 10)

d. 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. 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. 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. 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:

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