1. Explain the following different types of memories and their purposes:
a. Shared memory
b. Global memory
c. Constant memory

Answer:

a) Shared memory : shared memory stores variables differently than typical variables. It creates a copy of the variable for each block that you launch on the GPU. Every thread in that block shares the memory, but threads cannot see or modify the copy of this variable that is seen within other blocks. This provides a means by which threads within a block can communicate and collaborate on computations. Furthermore, shared memory buffers reside physically on the GPU. Because of this, the latency to access shared memory tends to be far lower than typical buffers, making shared memory effective as a per-block, software- managed cache.

b) Global memory : Data stored in global memory is visible to all threads within the application including the host, and lasts for the duration of the host allocation.

c) Constant memory: it is used for data that will not change over the course of a kernel execution and is read only. Using constants rather than global memory can reduce the required memory bandwidth, however, this performance gain can only be realized when a warp of threads read at same location.

2:

a.

The __syncthreads() command is a block level synchronization barrier. That means it is safe to be used when all threads in a block reach the barrier. It is also possible to use __syncthreads() in conditional code but only when all threads evaluate such code otherwise the execution is likely to hang or produce unintended side effects . __syncthreads() waits until all threads within the same block has reached the command and all threads within a warp - that means all warps that belongs to a thread block must reach the statement.

b.

This change to the kernel will actually cause the GPU to stop responding, forcing you to kill your program.
Under normal circumstances, divergent branches simply result in some threads remaining idle, while the other threads actually execute the instructions in the branch.
The CUDA Architecture guarantees that no thread will advance to an instruction beyond the __syncthreads() until every thread in the block has executed the __syncthreads().

Unfortunately, if the __syncthreads() sits in a divergent branch, some of the threads will never reach the __syncthreads(). Therefore, because of the guarantee that no instruction after a __syncthreads() can be executed before every thread has executed it, the hardware simply continues to wait for these threads. And ends up waiting forever.

This is the situation in the dot product example when we move the __syncthreads() call inside the if() block. Any thread with cache Index greater than or equal to it will never execute the __syncthreads(). This effectively hangs the processor because it results in the GPU waiting for something that will never happen.