Quiz Questions for Module 6

1. Assume that we want to use each thread to calculate two (adjacent) output elements of a vector addition. Assume that variable i should be initialized with the index for the first element to be processed by a thread. Which of the following should be used for such initialization to allow correct, coalesced memory accesses to these first elements in the following statement?

   ```
   if(i<n) C[i] = A[i] + B[i];
   ```

   (A) int i=(blockIdx.x*blockDim.x)*2 + threadIdx;
   (B) int i=(blockIdx.x*blockDim.x + threadIdx.x)*2;
   (C) int i=(threadIdx.x*blockDim.x)*2 + blockIdx.x;
   (D) int i=(threadIdx.x*blockDim.x + blockIdx.x)*2;

   Answer: (A)

   Explanation: Each thread block will process a section of each vector that contains twice as many elements as the number of threads in the thread block. We want to have all threads to process the first half of the section with each adjacent thread processing adjacent elements. The expression in (A) allows such a pattern. It also meets the judging criterion shown in Lecture 6.2.

2. Continuing from Question 1, what would be the correct statement for each thread to process the second element?

   (A) If (i<n) C[i+1] = A[i+1] + B[i+1];
   (B) If (i+1<n) C[i+1] = A[i+1] + B[i+1];
   (C) If (i+threadIdx.x < n) C[i+threadIdx.x] = A[i+threadIdx.x] + B[i+threadIdx.x];
   (D) if(i+blockDim.x < n) C[i+blockDim.x] = A[i+blockDim.x] + B[i+blockDim.x];

   Answer: (D)
   Explanation: all threads should be shifting to the next half of the section. Thus everyone should be moving to the element that is of BlockDim.x elements away from the first element it processed. The expression is really (blockIdx.x*blockDim.x)*2 + threadDim.x + threadIdx.x, which meets the judging criterion shown in Lecture 6.2.

3. Assume the following simple matrix multiplication kernel

```
__global__ void MatrixMulKernel(float* M, float* N, float* P, int Width)
{
  int Row = blockIdx.y*blockDim.y+threadIdx.y;
  int Col = blockIdx.x*blockDim.x+threadIdx.x;
  if ((Row < Width) && (Col < Width)) {
     float Pvalue = 0;
     for (int k = 0; k < Width; ++k) {Pvalue += M[Row*Width+k] * N[k*Width+Col];}
     P[Row*Width+Col] = Pvalue;
   }
 }
```

Based on the judging criterion in Lecture 6.2, which of the following is true?

(A) M[Row*Width+k] and N[k*Width+Col] are coalesced but P[Row*Width+Col] is not
(B) M[Row*Width+k], N[k*Width+Col] and P[Row*Width+Col] are all coalesced
(C) M[Row*Width+k] is not coalesced but N[k*Width+Col] and P[Row*Width+Col] both are
(D) M[Row*Width+k] is coalesced but N[k*Width+Col] andt P[Row*Width+Col] are not

Answer: (C)
Explanation: M is accessed with Row*Width+k, which is actually (blockIdx.y*blockDin.y+threadIdx.y)*Width + k where threadIdx.y has Width coefficient. This violates the criterion. On the other hand, N and P are accessed with k*Width+Col, which is actually (k*Width + blockIdx.x*blockDim.x)+threadIdx.x. This meets the judging criterion.

4. For the tiled single-precision matrix multiplication kernel in question 3, assume that each thread block is 32×32 and the system has a DRAM bust size of 128 bytes. How many DRAM bursts will be delivered to the processor as a result of loading one A-matrix element by a thread block (one k step)? Keep in mind that each single precision floating point number is four bytes.

(A) 16
(B) 32
(C) 64
(D) 128

Answer: (B)
Explanation. For 32×32 thread-block, all threads with the same threadIdx.y value will access the same A element. So, only 32 different A elements will be accessed, each accessed by a set of threads of a particular threadIdx.y value. However, each element belongs in a different DRAM burst so 32 bursts will be delivered to the processor.