Cache coherence:
1. May give you the transition diagrams for MSI (slides 20 and 21) and ask you to provide a similar transition diagrams for MESI.

Response to processor events:
- Read (RD)
- Write (W)

Response to Bus events:
- Read request for block B: BusRd (B),
- Write request for block B: BusRdX (x),
- Invalidate block B: BusInv (B)

Chapter 4:
4. For each of the following questions, you need to justify your answer.

Using vector instructions can you implement a “load with stride” operation using “load with gather” operation?

Using a vector instructions can you implement a “load with gather” operation using “load with stride” operation?
Which one of these sentences is true and which is false?

- All the threads of a thread block execute in lock-step  
  False
- 
  
  _syncthreads() is a barrier for all the threads in a thread block
  True
- Variable declared as _global_ in a CUDA kernel are allocated in the shared memory
  False
- Shared memory in CUDA is shared by all the threads in a kernel
  False
- Global memory in CUDA is shared by all the threads
  True
- cudaMemcpy() can be called from a Kernel to copy data between host and global memory
  False
- cudaMemcpy() is used to copy data between host and global memory
  True

Assuming that you wrote a cuda kernel that declares a shared memory array consisting of 4K bytes and that the compiler determined that each thread in that kernel needs 16 integer registers. Assume also that your GPU has 4 SMs, each with a register file of 2048 integer registers and a shared memory of 16K bytes. If your application will execute kernel

\[
\text{<<<nblocks, blksize>>>, answer the following questions:}
\]

- What is the maximum number of threads that can execute simultaneously on the GPU?
  
  Each SM has 2048 registers and each thread needs 16 registers
  \[ \rightarrow \text{each SM can support 128 threads} \rightarrow 4 \text{ SMs can support 512 threads} \]

- What is the maximum number of thread blocks that can execute simultaneously on an SM?
  
  Each SM has 16K bytes of shared memory and each thread block needs 4K bytes
  \[ \rightarrow \text{each SM can support at most 4 thread blocks simultaneously.} \]

- To execute the maximum number of threads simultaneously what is the value of nblocks and blksize that you would use when launching the kernel
  
  \[
  \text{Kernel} \lll 16, 32 \rrr \text{ or } \lll 8, 64 \rrr \text{ or } \lll 4, 128 \rrr
  \]
Show the output of the content of array A after the execution of the following program:

```c
_global_ F(int *A)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    A[idx] = idx;
    A[blockIdx.x] = blockIdx.x;
};
```

```c
void main()
{
    Allocate a 16 element int array A in the GPU global memory and initialize its elements to 0;
    F<<<2,4>>>(A);
}
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Show the output of the content of array A after the execution of the following program:

```c
_global_ F(int *A)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    A[row][col] = blockIdx.x + blockIdx.y + threadIdx.x;
};
```

```c
void main()
{
    Allocate a 6x6 array A in the GPU global memory;
    initialize A's elements to 0;
    dim3 grid(2,2); // a 2x2 array of blocks
    dim3 blocks(3,3); // each block is a 3x3 array of threads
    F<<<grid,blocks>>>(A);
}
```

```
<table>
<thead>
<tr>
<th>A(0,0)</th>
<th>A(0,1)</th>
<th>A(0,2)</th>
<th>A(0,3)</th>
<th>A(0,4)</th>
<th>A(0,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>A(1,0)</td>
<td>A(1,1)</td>
<td>A(1,2)</td>
<td>A(1,3)</td>
<td>A(1,4)</td>
<td>A(1,5)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>A(2,0)</td>
<td>A(2,1)</td>
<td>A(2,2)</td>
<td>A(2,3)</td>
<td>A(2,4)</td>
<td>A(2,5)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>A(3,0)</td>
<td>A(3,1)</td>
<td>A(3,2)</td>
<td>A(3,3)</td>
<td>A(3,4)</td>
<td>A(3,5)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>A(4,0)</td>
<td>A(4,1)</td>
<td>A(4,2)</td>
<td>A(4,3)</td>
<td>A(4,4)</td>
<td>A(4,5)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>A(5,0)</td>
<td>A(5,1)</td>
<td>A(5,2)</td>
<td>A(5,3)</td>
<td>A(5,4)</td>
<td>A(5,5)</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>
```
Show the output of the content of array A after the execution of the following program:

```c
_global_ F(int *A)
{
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    A[threadIdx.y][threadIdx.x] = blockIdx.x;
}

void main()
{
    Allocate an 6x6 array A in the GPU global memory;
    initialize A's elements to 0;
    dim3 grid(2, 2); // a 2x2 array of blocks
    dim3 blocks(3, 3); // each block is a 3x3 array of threads
    F<<<grid, blocks>>>(A);
}
```

Rewrite the following cuda kernel without using shared memory. The kernel adds n integers stored in the global array “input[]” into a global variable, “total”, and is called as `reduce<<<nb, n/nb>>>(input, n, total)` Where n is a multiple of nb.

```c
_global_void reduce(int *input, int n, int *total_sum)
{
    int tid = threadIdx.x;
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    for(int h=blockDim.x/2; h>0; h=h/2)
    {
        if (tid < h) x[tid] += x[tid + h];
        _syncthreads();
    }
    if (tid == 0) atomicAdd(total_sum, x[tid]);
}
```