ECE459 Final W2012

Question 1

1. Concurrency is where we manage multiple tasks at the same time, such as by interleaving tasks. The main programming concern is synchronization. Parallelism is the simultaneous execution of multiple tasks, ie on multiple cores. The main programming concern is efficient decomposition.
2. data is the variable involved in a dependency, it is a WAW (write after write) dependency. We could modify the code to use a different variable on the second write to store the result.
3. Not covered.

Question 2

Not covered, Gustafson's Law.

1. Since we have 20 seconds, half the time is sequential setup and half is parallelizable. Using Gustafson's Law, we have that speedup = 0.5 + 0.5*4 = 2.5x.
2. Using Gustafson's Law, we have that speedup = 0.01 + 0.99*4 = 3.97x.
3. The parallelizable fraction is 0.5. Using Amdahl's Law, we have that 3.97 = 1/(0.5+0.5/N). Solving for N, we get that it is negative. It is not possible to achieve such a speedup.

Question 3

1. Deadlock can occur. An execution that leads to deadlock is as follows.
    a. thread1 calls swap(a,b)
    b. thread2 calls swap(b,a)
    c. thread1 locks a->lock
    d. thread2 locks b->lock
    e. thread1 attempts to acquire b->lock, is blocked
    f. thread2 attempts to acquire a->lock, is blocked

2. We can fix this by enforcing a lock acquisition ordering. One approach could be to use the memory address as the ordering criterion. Locks are acquired in consistent order and released in reverse order.
3. The mutex locks ensure exclusive access to the data being modified. Consistent global ordering of lock acquisition prevents deadlock, since there is no circular wait.

Question 4

1. The outer loop should be parallelized, as each outer loop iteration computes one element of the out array independently; there is no dependency. Parallelizing the inner loop would require synchronization which adds overhead.
2. Not covered.

Question 5

Not covered.

Question 6

Everything is in CUDA instead of OpenCL.

1. read only: mat1, mat2, each a 2D array of size row x col
   write only: out, 2D array of size row x col
2. ((row*col)/256) + 1 blocks; 256 threads/block (2D indexed, 16x16)
3. __global__ void matAdd(double *mat1, double *mat2, double *out, int row, int col) {

```
        int j = blockIdx.x * blockDim.x + threadIdx.x;
        int i = blockIdx.y * blockDim.y + threadIdx.y;
        if (i < row && j < col) {
                out[i*col+j] = mat1[i*col+j] + mat2[i*col+j];
        }
}
```