CSCI-GA.3033-009
Multicore Processors: Architecture & Programming
Homework 3

1. There are several ways to deal with that, depending on the application at hand.
    - A lock is used to access a shared data. Then try to replicate this shared data. For example, if 100 threads want to update a data item, you can replicate this data item 10 times so that each 10 threads will be using a lock. Then you combine the 10 versions of the data item later.
    - Consider partitioning the resource and use a lock for each partition. For example, do not lock a full array, but partition this array into sub-arrays and use lock for each sub-partition.

2. x=1 x=2 x=3

3. Sources of performance loss in OpenMP:
    - Synchronization
    - Coherence
    - Communication
    - Memory access

4. A double float is 8 bytes of size. A cache block of 64 bytes can then carry 8 elements. With 8000 elements y will be partitioned (approximately) as follows
Thread 0: y[0], y[1], . . . , y[1999]
Thread 1: y[2000], y[2001], . . . , y[3999]
Thread 2: y[4000], y[4001], . . . , y[5999]
Thread 3: y[6000], y[6001], . . . , y[7999]
In order for false-sharing to occur between thread 0 and thread 2, there must be elements of y that belong to the same cache line, but are assigned to different threads. On thread 0, the cache line that's closest to the elements assigned to thread 2 is the line that contains y[1999]. But even if this is the first element of the cache line, the highest possible index for an element of y that belongs to this line is 2006:

y[1999] y[2000] y[2001] y[2002] y[2003] y[2004] y[2005] y[2006]

Since the least index of an element of y assigned to thread 2 is 4000, there can't possibly be a cache line that has elements belonging to both thread 0 and thread 2.

5. If we look at the location of y[0] in the first cache line containing all or part of y we see that y can be distributed across cache lines in eight different ways. If y[0] is the first element of the cache line, then we'll have the following assignment of y to cache lines:
first line y[0] y[1] y[2] y[3] y[4] y[5] y[6] y[7]
If y[0] is the second element of the cache line, then we'll have the following assignment:
first line _ y[0] y[1] y[2] y[3] y[4] y[5] y[6]
second line y[7] _ _ _ _ _ _ _
As a final example, if y[0] is the last element of the first line, then we'll have the following assignment
first line _ _ _ _ _ _ _ y[0]
second line y[1] y[2] y[3] y[4] y[5] y[6] y[7] _
(a) From our first example above, we see it's possible for y to fit into a single cache line.
(b) However, in most cases, y will be split across two cache lines.

6. (a)

1: $x = x + 1$;
2: $a = x + 2$;
3: $b = a + 3$;
4: $c = c + 1$;

As written, 1 must be executed before 2, and 2 must be executed before 3 because of true read after write dependencies. However 4 is completely independent so two concurrent threads could be used.

(b) If we re-write it as follows:

1: $x = x + 1$;
2: $a = x + 3$;
3: $b = x + 6$;
4: $c = c + 1$;

Then the 4 instructions are independent and 4 threads can be used.

7. Yes, if the overhead of parallelization (setting the runtime, communication, …) is larger than the performance gained due to parallelization.