1. Assume we are multiplying an 8000x8000 matrix with vector y. Each element of the matrix and vector is double float. Also suppose that thread 0 and thread 2 are assigned to two different cores. If a cache line contains 64 bytes. Is it possible for false sharing to occur at any time between threads 0 and 2?

2. Assume we have a dual-core processor and we are doing a matrix vector multiplication where the matrix is 8x8,000,000 and we multiply it by vector y. Each element is a double float (i.e. 8 bytes) and a cache line is 64 bytes. We parallelized this program using 4 threads.
   a. What is the minimum number of cache lines that are needed to store vector y?
   b. What is the maximum number of cache lines that are needed to store vector y?

3. Explain and justify what is the scope of the variables in the following block of code and identify eventual problems, if any.

   #pragma omp parallel for private(a,b)
   for (i = 0; i < N; i++) {
       int x = 0;
       c--;
       for (j = i; j < N; j++)
           x += func(c, b[j]);
       a[i] = x;
   }

4. Identify the dependencies loop-carried existent in the following code block, and write a parallel version of the code in OpenMP with the dependencies removed.

   for (i = 0; i < N - 2; i++) {
       a[i] += a[i + 2] + 5;
       x += a[i];
   }
5. As a CUDA programmer, how does knowing about the concept of warps help you, especially that warps are transparent to the programmer?

6. Suppose an NVIDIA GPU has 8 SMs. Each SM has 32 SPs, but a single warp is only 16 threads. The GPU is to be used to add two arrays element-wise. Assume that the number of array elements is $2^{24}$. Let $t$ denote the amount of time it takes one thread (yes, just one) to perform the entire calculation on the GPU. The kernel code is shown below (num_threads is the total number of threads in the whole GPU):

```c
__device__ void prob(int array_size) {
    int tid = threadIdx.x + blockIdx.x * blockDim.x;
    for ( int i=tid; i<array_size; i += num_threads )
        result[i] = a[i] + b[i];
}
```

(a) What is the amount of time it takes if we use one block of 16 threads?
(b) What is the amount of time it takes if we use two blocks of 8 threads each?
(c) Justify why the above two answers are similar/different.
(d) Assume that 256 threads are enough to keep all SPs in the SM busy all the time. What is the amount of time it would take to perform the computation for one block of 1024 threads? Justify.
(e) Repeat question (d) above but with two blocks of 512 threads each.

7. The line of code below checks for a special case to avoid calling an expensive square root. Describe a situation in which it makes sense for CUDA to do that, and a different situation when it makes no sense (meaning it would be faster to do the square root all the time). Assume that 50% of the time $d$ is equal to 1.

```c
if ( d == 1 ) s = 1; else s = sqrt(d);
```

8. Why do we still to deal with race condition even though the hardware in most multicore processors supports coherence?

9. State all reasons you can think of to decide whether to use MPI or OpenMP to solve a specific problem.

10. How many threads could be used for the computation below, each thread executing one or more of the instructions:
x++;
    a = x + 2;
    b = a + 3;
    c++;

without changing the code. Explain clearly your answer.