Download the source file *pth_ mat_vect_ rand_ split.c* from the book's website, or our course website, then complete the exercise 4.16 - 4.18 of Chapter 4.

4.16. Recall the matrix-vector multiplication example with the 8000×8000 input. Suppose that the program is run with four threads, and thread 0 and thread 2 are assigned to different processors. If a cache line contains 64 bytes or eight `doubles`, is it possible for false sharing between threads 0 and 2 to occur for any part of the vector `y`? Why? What about if thread 0 and thread 3 are assigned to different processors–is it possible for false sharing to occur between them for any part of `y`?

4.17. Recall the matrix-vector multiplication example with an $8 \times 8{,}000{,}000$ matrix. Suppose that `doubles` use 8 bytes of memory and that a cache line is 64 bytes. Also suppose that our system consists of two dual-core processors.

a. What is the minimum number of cache lines that are needed to store the vector `y`?

b. What is the maximum number of cache lines that are needed to store the vector `y`?

c. If the boundaries of cache lines always coincide with the boundaries of 8-byte `doubles`, in how many different ways can the components of `y` be assigned to cache lines?

d. If we only consider which pairs of threads share a processor, in how many different ways can four threads be assigned to the processors in our computer? Here we're assuming that cores on the same processor share a cache.

e. Is there an assignment of components to cache lines and threads to processors that will result in no falses sharing in our example? In other words, is it possible that the threads assigned to one processor will have their components of `y` in one cache line, and the threads assigned to the other processor will have their components in a different cache line?

f. How many assignments of components to cache lines and threads to processors are there?

g. Of these assignments, how many will result in no false sharing?

4.18. **a.** Modify the matrix-vector multiplication program so that it pads the vector `y` when there's a possibility of false sharing. The padding should be done so that if the threads execute in lock-step, there's no possibility that a single cache line containing an element of `y` will be shared by two or more threads. Suppose, for example, that a cache line stores eight `double` s and we run the program with four threads. If we allocate storage for at least 48 `doubles` in `y`, then, on each pass through the `for i` loop, there's no possibility that two threads will simultaneously access the same cache line.

b. Modify the matrix-vector multiplication so that each thread uses private storage for its part of `y` during the `for i` loop. When a thread is done computing its part of `y`, it should copy its private storage into the shared variable.

c. How does the performance of these two alternatives compare to the original program? How do they compare to each other?