5.10. Recall that all structured blocks modified by an unnamed `critical` directive form a single critical section. What happens if we have a number of `atomic` directives in which different variables are being modified? Are they all treated as a single critical section?

We can write a small program that tries to determine this. The idea is to have all the threads simultaneously execute something like the following code

```
   int i;
   double my_sum = 0.0;
   for (i = 0; i < n; i++)
#     pragma omp atomic
      my_sum += sin(i);
```

We can do this by modifying the code by a `parallel` directive:

```
#  pragma omp parallel num_threads(thread_count)
   {
      int i;
      double my_sum = 0.0;
      for (i = 0; i < n; i++)
#        pragma omp atomic
         my_sum += sin(i);
   }
```

Note that since `my_sum` and `i` are declared in the `parallel` block, each thread has its own private copy. Now if we time this code for large n when `thread_count` $= 1$ and we also time it when `thread_count` > 1, then as long as `thread_count` is less than the number of available cores, the run-time for the single-threaded run should be roughly the same as the time for the multithreaded run if the different threads' executions of `my_sum += sin(i)` are treated as different critical sections. On the other hand, if the different executions of `my_sum += sin(i)` are all treated as a single critical section, the multithreaded run should be much slower than the single-threaded run. Write an OpenMP program that implements this test. Does your implementation of OpenMP allow simultaneous execution of updates to different variables when the updates are protected by `atomic` directives?

5.11. Recall that in C, a function that takes a two-dimensional array argument must specify the number of columns in the argument list, so it is quite common for C programmers to only use one-dimensional arrays, and to write explicit code for converting pairs of subscripts into a single dimension. Modify the OpenMP matrix-vector multiplication so that it uses a one-dimensional array for the matrix.

5.12. Download the source file `omp_mat_vect_rand_split.c` from the book's website. Find a program that does cache profiling (e.g., Valgrind [49]) and compile the program according to the instructions in the cache profiler documentation. (For example, with Valgrind you will want a symbol table and full optimization. (With `gcc` use, `gcc -g -O2 . . .`). Now run the program according to the instructions in the cache profiler documentation, using input $k \times (k \cdot 10^6)$,

$(k \cdot 10^3) \times (k \cdot 10^3)$, and $(k \cdot 10^6) \times k$. Choose k so large that the number of level 2 cache misses is of the order 10^6 for at least one of the input sets of data.

- **a.** How many level 1 cache write-misses occur with each of the three inputs?
- **b.** How many level 2 cache write-misses occur with each of the three inputs?
- **c.** Where do most of the write-misses occur? For which input data does the program have the most write-misses? Can you explain why?
- **d.** How many level 1 cache read-misses occur with each of the three inputs?
- **e.** How many level 2 cache read-misses occur with each of the three inputs?
- **f.** Where do most of the read-misses occur? For which input data does the program have the most read-misses? Can you explain why?
- **g.** Run the program with each of the three inputs, but without using the cache profiler. With which input is the program the fastest? With which input is the program the slowest? Can your observations about cache misses help explain the differences? How?

5.13. Recall the matrix-vector multiplication example with the 8000×8000 input. Suppose that thread 0 and thread 2 are assigned to different processors. If a cache line contains 64 bytes or 8 `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`?

5.14. 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 cache.
- **e.** Is there an assignment of components to cache lines and threads to processors that will result in no false-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?