Teacher Instructions

Pose the problem to the students, and describe how the Sieve algorithm works. There are many different examples out there, including:

https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

http://www.shodor.org/petascale/materials/UPModules/sieveOfEratosthenes/

Introduce the students to the starter code. Note that the Sieve algorithm is a classic one for predicting lists of contiguous primes, and is already very fast. As they discuss running times and hypothesize how parallelization will improve the code, have them keep in mind realistic expectations. Codes that run in milliseconds, including all overhead and I/O, are not likely to show speedup in parallel.

Note the use of the OpenMP specific timers. Remind the students that standard clock timers that they might use to profile serial codes often work unexpectedly in parallel codes--leading to each of the different parallel programming libraries to typically implement their own timers. If students have already performed timing using other methods, have them compare and contrast OpenMP's omp_get_wtime method with other ways they have learned.

As they discuss the loop carried dependencies, they should see that each pass through the outer loop eliminates more sections within the inner loop. While technically this can be done concurrently without producing wrong output, much of the speedup of the algorithm comes from being able to skip numbers that have already been determined not prime--thus parallelizing the outer loop does carry a loop carried dependency. The inner loop does not, however students may see efficiency loss due to many threads accessing nearby elements of memory in the variable "list." If students have already discussed false sharing, you might note for them the possibility of false sharing in the parallelization of the inner loop.

The exact values of N for which the students will see improvement in parallel is machine dependent, but you should expect that value to be large, approaching the value of MAX_INT on most machines. The starter code has thus been programmed to use longs instead of ints. As N is made larger, depending on compiler and machine you may see warnings that you are allocating large arrays, or you may fail to allocate enough memory to solve the problem. This also will be highly machine and compiler dependent, so practice what the students will do ahead of time.

When the students have completed the activity, they should be able to run performance testing, and they will see that for small N the code does not benefit and is perhaps worsened by parallelism, but that as N grows the benefit of parallelism can be seen. They should not expect to see strong scaling for this example, as this is an efficient $n \log(n)$ serial algorithm already, and by the time the problem has grown to a size where parallelism is useful they will also likely be approaching machine and memory limits.