In this homework on Gaussian elimination. I chose to parallelize the row and col loop but not the first loop (norm loop) because on the first loop, the calculation of the norm+1 entries depend on the operations done at previous values of norm. So we cannot parallelize the first loop.

In OpenMP, I chose to use

#pragma omp parallel for private(multiplier)

Because it is the simplest implementation. I put the variable multiplier in the private directory because for each thread the multiplier variable is used privately depending on which row we are in.

In Pthread, this is what I did:
-Made a struct called arg_struct to carry 2 data points I need to plug into 4th argument in pthread_create: norm and threadid/threadnumber (in my code I just call it 'thread' for simplicity). Since we are not parallelizing the first loop, the 'norm' variable has to be remembered somehow thus the reason of arg_struct. 'thread' is just a variable to decide which thread is doing which part of the loop.

-in gauss(), I create pthread_t according to how many threads we put in the command line argument (default number of threads is 4)

-I then basically divide the for loop into (by default) 4 parts and let 4 threads execute them, also passing in an arg_struct so that each thread knows which part of the loop its doing, and on which norm position.

-finally I join the threads.

This implementation is straightforward and effective. The more threads we give, the faster the program runs. We can try that with different command line arguments.

In OpenMP, I think in the future I can optimize the code more by using OpenMP API, meaning I could manually divide the work between threads just like pthread.