Name: Redon Jashari

Assignment #4

Problem 4.1: **multi-threaded coin flipping**

I have an Intel Core i7-10750H with 12 cores and 16 GB of Memory.
After running my program, these are the results I got:

coins: 0000000000XXXXXXXXXX (start - global lock)
coins: 0000000000XXXXXXXXXX (end - global lock)
100 threads x 10000 flips: 79.735 ms

coins: 0000000000XXXXXXXXXX (start - iteration lock)
coins: 0000000000XXXXXXXXXX (end - iteration lock)
100 threads x 10000 flips: 5851.017 ms

coins: 0000000000XXXXXXXXXX (start - coin lock)
coins: 0000000000XXXXXXXXXX (end - coin lock)
100 threads x 10000 flips: 9834.865 ms

Strategy 1:

Here we can see that the first strategy performs much faster than the other strategies because:

1. It has minimal synchronization overhead; each thread acquires the lock/unlock only once, so it eliminates repeated cost.

2. While a thread holds the lock, it repeatedly touches the same memory with no other core interfering.

Strategy 2:

For this strategy, the reasons it performs much more slowly are:

1. Since each thread performs N lock/unlock on the same mutex, which causes a huge number of contented lock ops

2. The lock/unlock cost is higher than the flip cost, where the overhead of getting the lock repeatedly dominates runtime.

Strategy 3:

The third strategy is by far the worst since:

1. Even more lock/unlock operations. Instead of 1 or N locks per thread, you have 20 x N locks per thread.

2. Each coin mutex is likely a separate cache line; tens or hundreds of threads repeatedly grabbing different mutexes causes heavy coherence traffic

3. You have 100 threads, but only 20 coins. Many threads are fighting over the same coin locks, which causes a high probability of contention per coin.