Jayendra Jog

904437296

jayendra@ucla.edu

Yu-Kuan (Anthony) Lai

004445644

yukuan.anthony.lai@gmail.com

**1.1**

===================================================

1. It generally takes (roughly) greater than 5000 total operations for

an error to occur consistently. It takes this many threads and iterations

because the race conditions are not accounted for in the add function.

It is possible for the value of pointer to change incorrectly if the

scheduler gives control to another thread in the middle of the add

function. For example, if one thread has sum = 5 + 1 = 6, and then

control passes to another thread, which runs 100 iterations of

sum = 5 + 1 => 105, and then sets pointer = 105, then when the first

thread regains control, it will set pointer = 6, effectively making the

100 additions from the second thread useless. This showcases how a large

number of operations causes problems. Moreover, with more iterations and

threads, there are more chances for these race conditions to come about.

2. With less iterations, there are significantly less chances for the

race conditions to come about. Moreover, with less iterations, each

thread has less time when its being run, so there is a lesser chance for

the scheduler to interrupt it.

**1.2**

===================================================

1. The average cost per operation drops because there is a high overhead

for the allocation of memory, creating and execution of threads, and the

ultimate joining of threads.

2. We can know the correct cost of implementation by seeing what the cost/op

converges towards as the number of iterations approaches infinity. When the

iterations becomes very high, then the cost due to overhead becomes

insignificant, so it represents a much more accurate and "correct" cost per

operation.

3. When yield gets called, a system call causes the kernel to receive control

and then pass control over to the relevant thread. Passing control over to the

kernel is a slow process, and doing this for every single interation of every

single thread accounts for the slowdown.

4. Using --yield does not allow us to get valid timings. The yield function

gives control to the CPU for a variable amount of time, and thus it becomes

impossible to get an exact time calculation. Moreover, each yield call accounts

for even more time, so the yield calls in themselves will skew the cost/op to

increase, even though the operation doesn't actually need all of that time.

**1.3**

===================================================

1. All of the options perform similarly for low numbers of threads because

the computation of locks for a low number of threads doesn't require too much

time. Essentially, the price per lock is very cheap when there aren't too many

threads, so the time increase due to the lock is relatively insignificant.

2. As the number of threads increases, there are more and more threads competing

for the same resource (the lock). This requires more kernel calls to figure out

which thread should get access, and the end result is that with more threads,

there is a greater amount of time needed.

3. Spin locks spend a lot of time waiting, so they are extremeley inefficient for

a large number of threads. The more threads there are, the longer each spin lock

takes, which is why there is a net increase in the amount of time when the number

of threads increase.