Jason Zhang

Parallel Written HW2

1

a.

In this particular case since we are sharing one Random object between all my threads, when they try to generate another random int , it needs to call the synchronized method “next”. Since “next” is synchronized, that means that only one thread can access and use that method at a time and it locks out that method from being used by other threads until it is done. In our program since each thread needs to repeatedly get a random int until the end of a for loop, each thread will lock out this “next” method from the other threads each time it tries to get a random int which makes it nearly a serial execution due to all of the locking. Thus, we expect CONSTANT speedup.

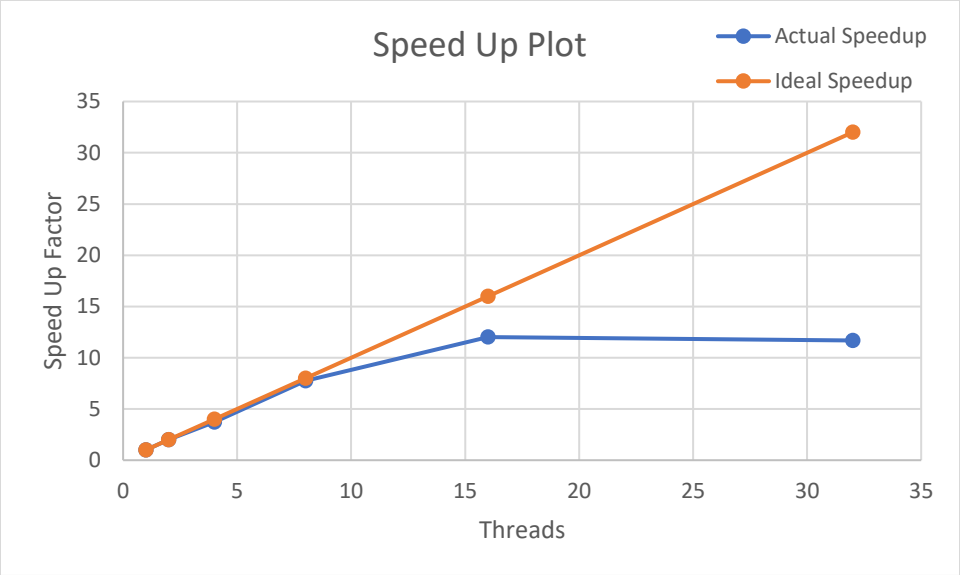
1

b.

The factor against parallelism it demonstrates is interference because of the synchronized next method. All of the threads need to access the shared resource of the random object and at that point we will have a conflict of access between threads which will drastically slow down our execution because our threads will have to wait until the next method is not in use by any other thread.

2

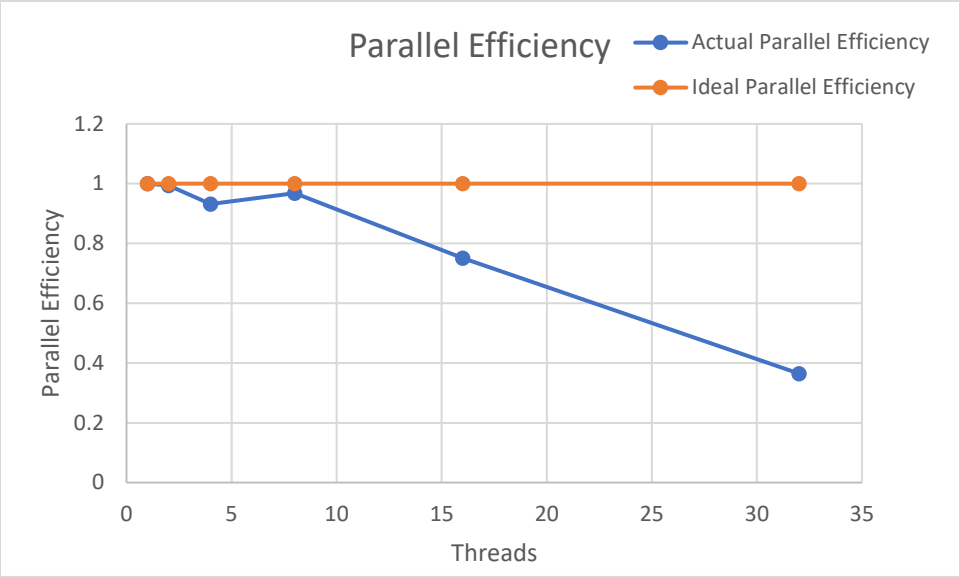
a.


Speed Up Plot
Actual Speedup
Ideal Speedup
Speed Up Factor
Threads
0
5
10
15
20
25
30
35

2 a. continued

I characterize this as a sub linear plot because the speed up starts to level off at around 8 to 16 cores. This is most likely due system restrictions on AWS. We are given a 16-virtual core instance which doesn't necessarily mean 16 separate CPU cores but rather possibly fewer cores that are hyper-threaded. Hyperthreading gives us two "logical cores" per CPU which makes it seem like there are two cores per CPU. This gives us the pro that we can have parallel execution of the two logical cores within only a single CPU unit. Of course, the downside is that both logical cores need to share the same execution resources which include the cache line. This can be especially bad if both logical cores are running processes that try to access the cache. In this case we will run into many cache misses and slow down both logical cores. Clearly the two hyperthreaded logical cores aren't as good as two separate cores, since the separate cores don't need to share execution resources. From this hardware restriction we should see our speed up deviate a bit from the ideal. An obvious point to make is that past 16 threads we start to see no improvements to speed up and in fact maybe a slowdown due to the overhead of starting more threads than our system is equipped to handle, which is at most 16 threads. Also, the deviation at around 16 threads is likely due to us using 16 hyperthreaded cores since we don't have 16 separate cores. Hyper threaded cores are typically slower than distinct cores due to the resource sharing and thus we have our explanation for our deviation at around 16 threads.

2b.


Parallel Efficiency
Actual Parallel Efficiency
Ideal Parallel Efficiency
Parallel Efficiency
1.2
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20
25
30
35
Threads

2b. continued

I would characterize this as sublinear because our actual parallel efficiency starts to deviate from our ideal and gains a negative slope in comparison to the straight line of our ideal parallel efficiency. For up until 8 threads the parallel efficiency is still close to 1 meaning that we still get our ideal parallelized execution for those threads. It's not perfectly 1 most likely due to some start-up costs that slow down our execution. But once we get past 8 threads and go to 16 and beyond, we start to see some massive drop offs in parallel efficiency. This is due to our AWS EC2 system restrictions. Our actual system has 16 vCPUs, which means we have 8 actual CPUs, each are hyper threaded to give us 16 logical cores. This was meticulously research and tested for. Since we have 8 separate CPU cores, we can achieve near perfect parallel execution since we don't have the messy interference from the hyperthreaded cores needing to share execution resources (because we can schedule our 8 threads on each of the separate CPU cores). However, when it gets to beyond 8 threads, we need to start utilizing our hyperthreaded logical cores. These cores need to share execution resources which include the cache line. Thus, there can be problems of interference when the two hyperthreaded cores on the same CPU need to use the same resource at the same time which will lead to a slowdown in each thread's execution. An example would be if there's a need to access the cache at the same time, both threads will try to access, read, write, etc. the cache which will lead to cache thrashing, which is when a high number of cache misses occur due to both threads trying to modify the cache at the same time.