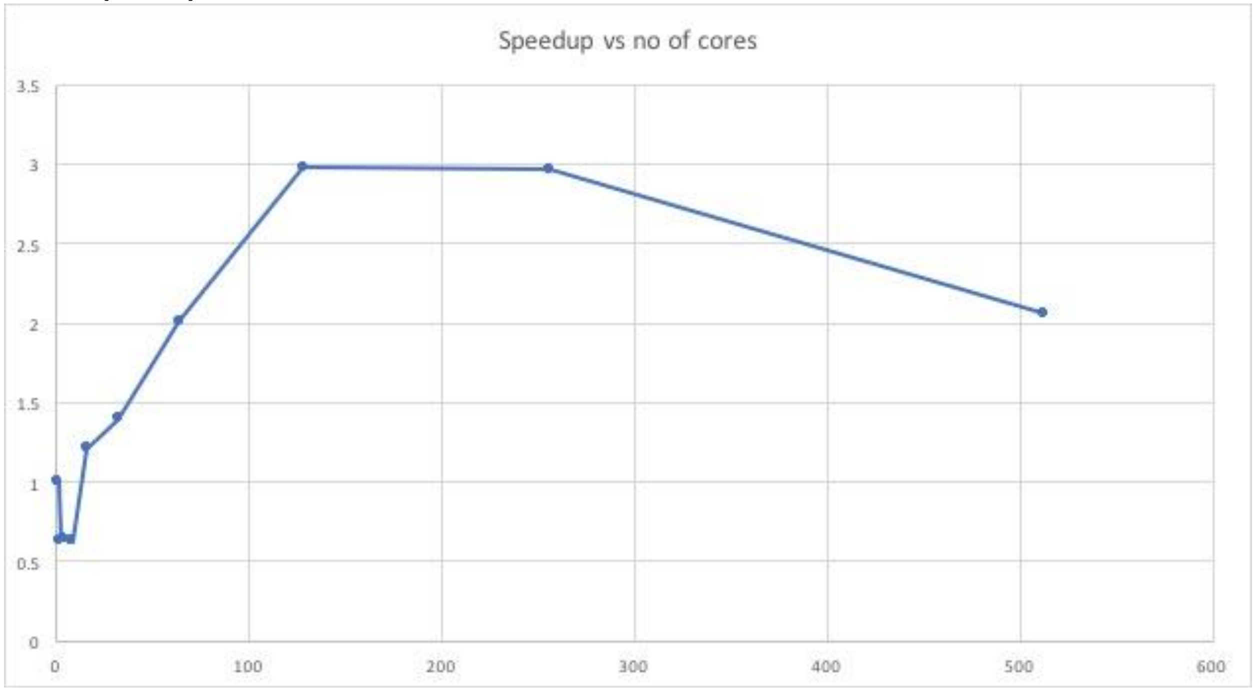
Q1 :
A)The souce code object is a synchronized object. That means that access to this object from each thread will be synchronized or will be one at a time. So this will mean that this part of the code is serial and since finding a random number is a step on each coin flip on each thread, this will lead to the whole program become serial. Factors against parallism = Interference

Also there will be a slowing of the program due to overheads associated to synchronization which are large.

B) There will be a speedup of close to 1 or less than 1. This will lead to a sublinear speedup with S ideal ~ 1.

Q2.

A) **X axis Speedup**



Yaxis : no of threads

B) There are parts of program which are serial. When all the coins are flipped, the total no of heads received is a serial implementation which takes up much computing time. Also there are startup costs associated to distribute task in threads etc which takes up resourses. Due to these 2 reasons, the code gives a sublinear speedup

C) It does increase till 128 threads as each of the 16 cores have some multithreading capacity (62EC2)which is more than the number of cores but degrades after that. That is because the task is being divided into as many threads are requested, which has startup costs associated with it, but there is no more new cores to do the computation faster. The commutating cores are the same and startup costs is increasing, this result in a degrading speedup curve. Bad !!

Q3.
A)The experiment is as follows : I will not run the coin flipping part of the code i.e taking random number and checking on so much cases imitating the coin flips.
This will only measure the time taken to distribute threads and distribute computations into it and the serial part of the code, whereas the computation associated with flipping coins will be negligible,. So I can directly calculate the time wasted in startup cost.

B)Startup cost for 1 thread run with 1000000000 coin Flips ~ 4600 Milli Sec.

The cost of full calculation is ~14600. SO it is less that the whole cost which makes sense. Also the algorithm makes it logical.

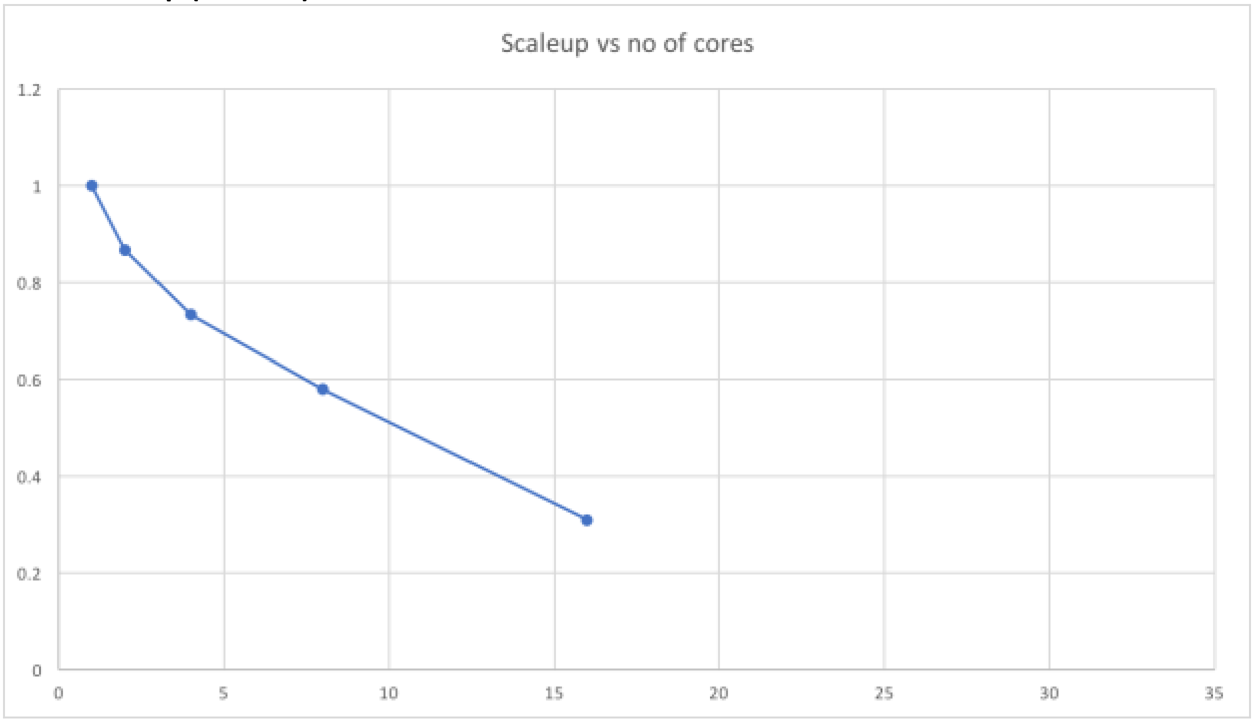
> B)   = p ~1-(4600/14600)
> S=1/(1-p+p/s)
> S=100 S= 3.10683
> S=500 S=3.16064
> S=1000 S=3.1675

Q2.
A)
The scaleup is for 1 thread -> 20 bit
2therads -: 21 bit and so on.

The result is a scaleupwhich is exponentially decreasing. As the problem size increases, the overhead and startup costs lead to a weak linear scaling in the scaleup.

Scaleup is TS/TL where TS-> small task time and TL is large task time by the computation with equal work on each core. i.e double the task, double the number of cores.

Xaxis Scaleup (no unit)



YAXIS : No of cores :

b)