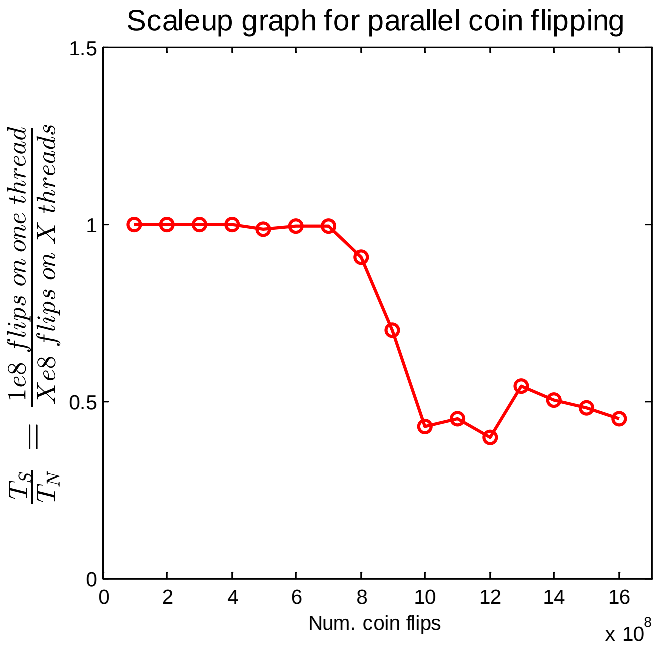
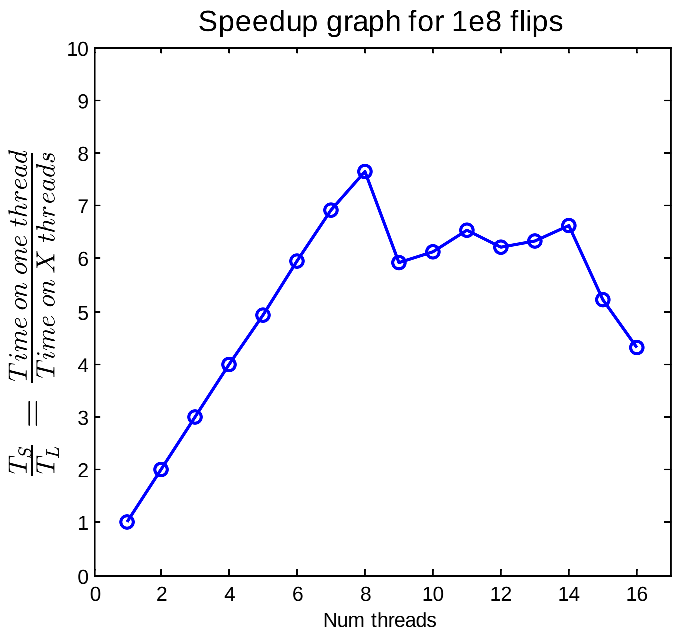
Kyle Mercer
3/7/2013
600.320

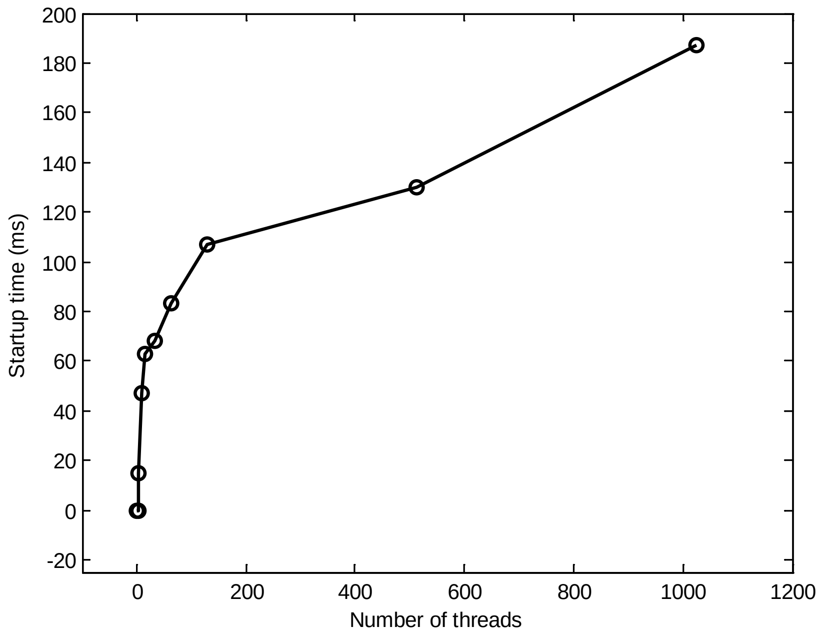
Assignment 2 – Write-up

1. Parallel coin flipping.
   a. See graphs 1 & 2
   b. For the scale-up graph (graph 1) we see a constant relation between T_S/T_N vs. number of coin flips up until about 8 cores/threads. Increasing the number of threads past the number of available core on the benchmark machine (AWS running on 8 cores) causes a loss in scale-up. We have a very similar effect for the speedup graph (graph 2). There is a loss of speedup after 8 cores. The sources of loss are attributed to the number of threads being greater than the number of available cores on the system. For scale-up, we are also increasing the problem size without the resources to increase the number of threads run concurrently. As such, after the number of threads exceeds the number of cores, there is massive loss in scale-up. In terms of speedup, the same phenomenon applies; that is, more than one thread is being run on each core after 8 threads are created, thus the processing power of each core is being divided across multiple threads cause our speedup to degrade. Perhaps an overload of context switch is occurring cause an even greater slowdown.





2. Startup cost experiment



a. The measurement of startup cost was done in *ParallelCoinFlipStartupCost.java*. The start time was recorded at the beginning of the program after some initial tests were done to "warm up" the JVM. The warm up portion was added to allow the overhead of the JVM to settle prior to measuring the startup cost. The end time was taken immediately following the *for loop* which starts each thread-object. This measures the startup time primarily because it is the portion of the code which cannot be parallelized. This includes, the calculation of flips per thread and the time to create each thread object and subsequently launch.
b. The estimated startup cost for a wide range of threads can be visualized in graph 3. The startup cost increases as the number of threads to launch increases which makes sense.
c. Using Amdahl's Law: $\frac{1}{(1-P)+\frac{P}{S}}$ we can calculate a P value using our speedup graph for any number of threads (say 4) with a corresponding speedup of:

$$3.98 = \frac{1616ms}{402ms} = \frac{T_1}{T_4}$$

P=0.984	
Threads	Speedup
100	38.69969
500	55.6545
1000	58.87894

For 4 threads, the startup time was 15ms (see startup graph). As such, the parallelized portion of the code, P, is:

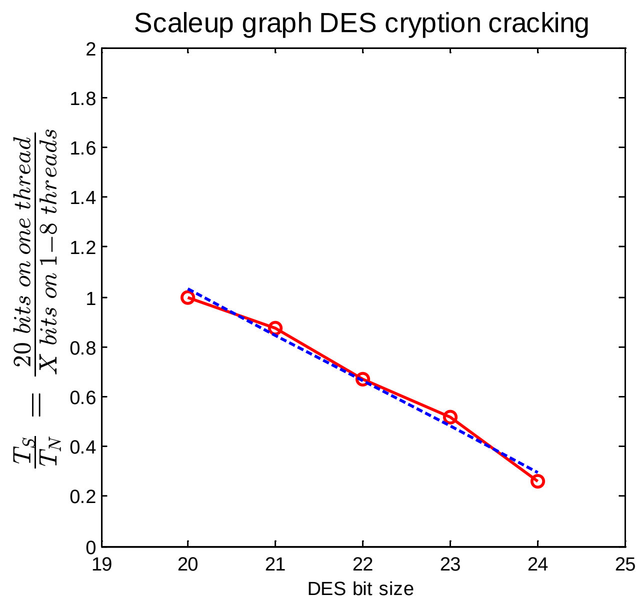
$$P = \frac{402ms}{402ms + 15ms} = 0.964$$

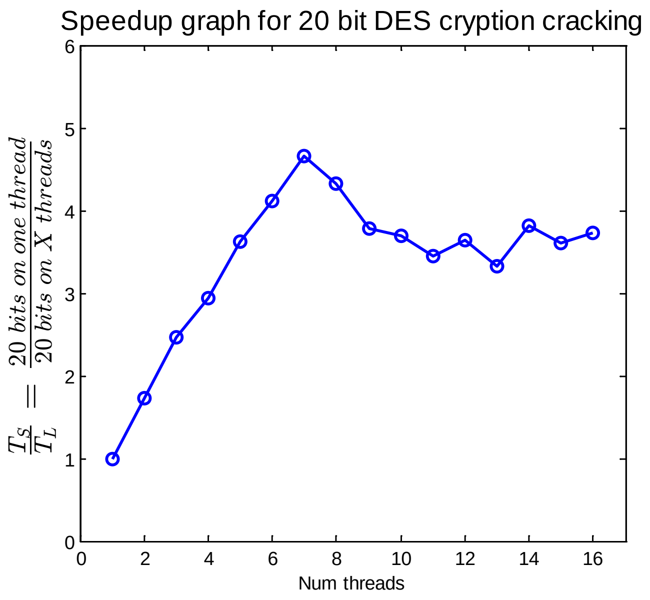
The table to the left shows the speedup for various thread counts based upon the relationship:

$$Speedup = \frac{1}{1 - 0.964 + \frac{0.964}{S}}$$

Where S is the number of threads to start up.

3. Parallelized DES encrypted message cracking





a. See graphs 4 &5 for the scale up and speedup respectively. The scale up is sub-linear (negative relation) whereas the speedup is linear up until the number of threads reaches the number available cores on the system.
b. Each time the bit size increases by one we are doubling the size of problem set. This is an exponential increase on the task size. Additionally, there are more serial operations inside the parallelized portions of the code (ie. setting the key, decrypting of the key, string comparison, etc.) which would add more interference across the shared memory sections of the cache (not that there was shared variables, but more memories in the shared region would get cycled more frequently, such as context switched).
c. Extrapolating the scale up graph to a power regression approximation yields the function:

$$\frac{T_S}{T_N} = 6E10x^{-8.321} = 0.1001, x = 26$$

As such:

$$\frac{1}{T_{26}} = \frac{0.1001}{9097ms} \rightarrow T_{26} = 89409ms = 90s$$

This is the amount of time to decrypt 26 bits on 64 cores. Thus, for every bit we increase we must double the time it takes. Thus the decryption would take:

$$T_{26} * 2^{30} = 96003107417s \approx 3044.23\ years$$