Parallel Programming
Assignment #2: Measuring Parallel Performance

Yifan Ge

March 8, 2013

Problem 1

For reasonable parameters, measure the scale-up and speed-up of this program from 1 thread to twice as many threads as cores. Your experiment should take seconds (so that it's a significant number of trials) but not hours. For example, for 8 cores, you might run the program for $1, 2, 3, \ldots, 16$ threads at 1000000000 coin tosses for speedup. (Don't worry about +/- rounding errors for number of flips per thread.) And, run the program for $1, 2, 3, \ldots, 16$ threads at $1000000000, 2000000000, \ldots, 16000000000$ for scaleup.

Scaleup and speedup

1. **Produce charts that show the scaleup and speedup of your program.**
   The speedup of the program is shown in Figure 1, and the scaleup is shown in Figure 2.

2. **Algorithm (true) speedup/scaleup measures the scaling performance of the algorithm as a function of processing elements. In this case, from $1 \ldots 8$. Characterize the algorithmic speedup/scaleup. If it is sub-linear, describe the potential sources of loss.**
   When the number of threads ranging from $1 \ldots 8$, both the speedup and scaleup are almost linear. Only when the number of threads reaches 8, the speedup/scaleup start to degrade. This is because I am running the program on a 8-core machine, so the optimal number of threads is 8. And the JVM is running a master thread, so the speedup and scaleup cannot keep linear when the number of slave threads reaches 8.

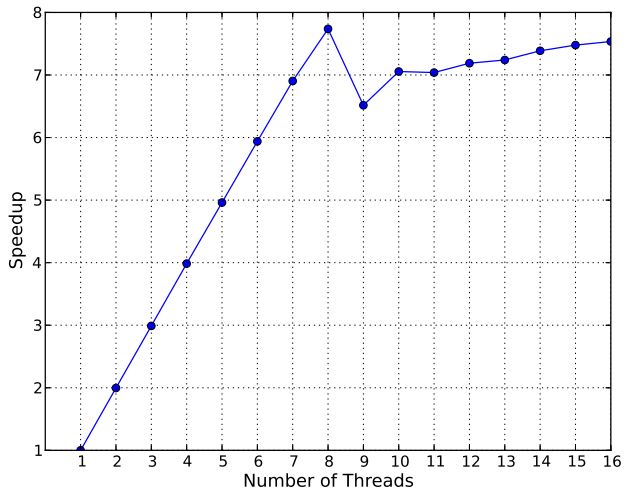

Figure 1: Speedup of CoinFlip for different number of threads

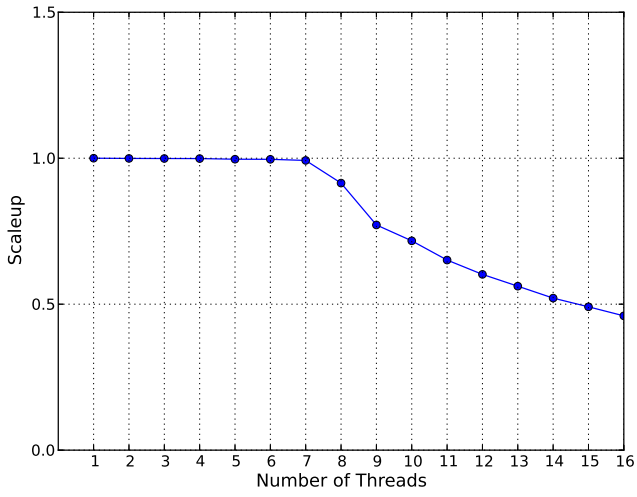

Figure 2: Scaleup of CoinFlip for different number of threads

3. Why does the speedup not continue to increase past the number of cores? Does it degrade? Why?
   As the processor has 8 cores, it can have at most 8 instructions executed per cycle. When I have 8 threads running, the processor's execution resources have been well utilized. This is why the speedup does not continue to increase past the number of core. An obvious degradation can be observed when there are 9 threads running. This is because of the latency of thread switching and startup cost.

Design and run an experiment that measures the startup costs of this code.

1. **Describe your experiment. Why does it measure startup?**
   If we run the program with 0 coin flippings, the time we get is the startup cost. This is because the threads are created and initialized but never actually put to work.

2. **Estimate startup cost. Justify your answer.**
   We run the program for $100, 200,$
   $\ldots, 500$ threads with 0 iteration. The result is shown in Figure 3.

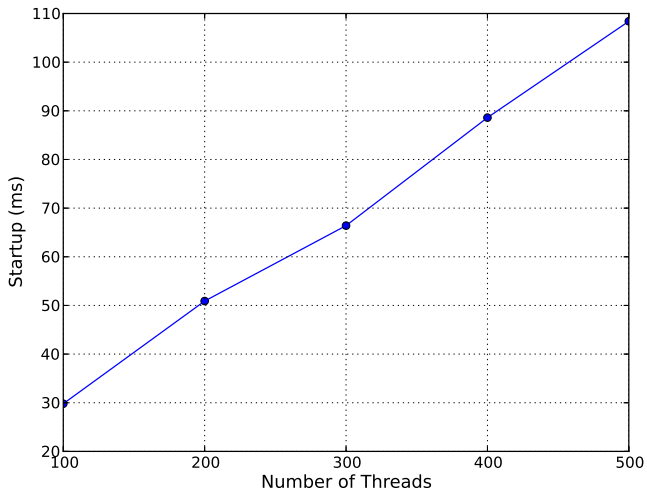

Figure 3: Startup of CoinFlip for different number of threads

From the figure, we estimate the startup cost $startup = 0.195 * \dot{\#}thread + 10.5$

3. **Assuming that the startup costs are the serial portion of the code and the remaining time is the parallel portion of the code,what speedup would you expect to realize on 100 threads? 500 threads? 1000 threads? (Use Amdahl's law.)**
   Assume the $\#thread$ is 1, the startup cost is about 10.7 ms, and from part 1 we know the running time is 9050 ms, so we estimate the parallel portion to be $\frac{9050-10.7}{9050} = 0.99$. Using Amdahl's law, the speedup is $\frac{1}{(1-P)+\frac{P}{S}}$. So with $P = 0.99$, for $S = 100$, the speedup is 50.25. For $S = 500$, the speedup is 83.47, and for $S = 1000$, the speedup is 90.99.

Problem 2

For reasonable parameters and for however many cores you have on the system, measure the scaleup and speedup of this program.

1. **Produce charts and interpret/describe the results. Is the speedup linear?**
   The program is run for $1, 2, \ldots, 16$ threads with a key of 20 bits long for speedup, and run for $1, 2, 4, 8, 16$ threads with $20, 21, 22, 23, 24$ bits long keys for scaleup. The speedup of the program is shown in Figure 4, and the scaleup is shown in Figure 5.

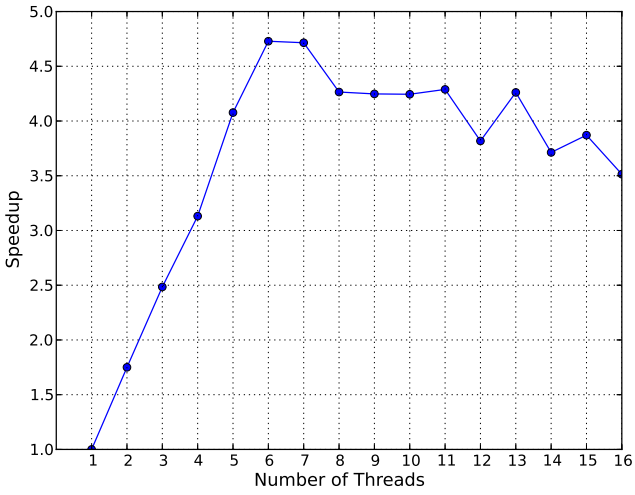

Figure 4: Speedup of BruteForceDES for different number of threads

   The speedup shows a near-linear increase when the thread number goes from 1 to 6, but it is not a linear speedup as doubling the number of threads doesn't double the speed. When the thread number reaches 7, the increase gets down and finally starts to degrade.
   The scaleup keeps going down. And from 1 thread to 2 threads, the drop is particularly rapid. It is also not linear scaleup.

2. **Why do you think that your scaleup/speedup are less than linear? What are the causes for the loss of parallel efficiency?**
   As I use a separate `SealedObject` for each of the working threads, and I make every thread create its own `SealedDES` object to do the decryption,

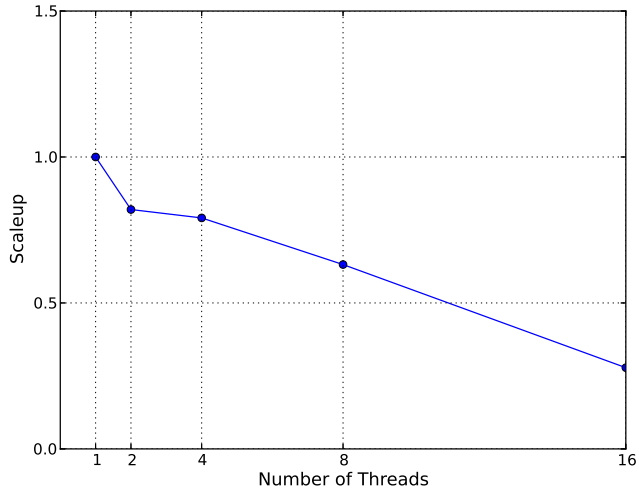

Figure 5: Scaleup of BruteForceDES for different number of threads

there should be no shared resources between the threads. And as I have less threads than cores, there should be not much context switching overhead to affect the speedup. And the job is divided evenly among all the threads, there is no skew problem. So the cause of the sublinear speedup/scaleup is the startup cost. The portion of operations that cannot be run in parallel has lower the speedup increase, and because of Amdahl's law the speedup and scaleup are sublinear.

3. **Extrapolating from your scaleup analysis, how long would it take to brute force a 56 bit DES key on a machine with 64 cores? Explain your answer.**
   Assume the scaleup to be S_{64} with 64 threads. We suppose for 1 thread and 20-bits key, the running time is T_1; for 64 threads and 26-bits key, the running time is T_{64}. Here we have $S_{64} = T_1/T_{64}$. From previous analysis, $T_1 = 10051ms$. To brute force a 56 bit DES key, the time should be $T = 2^{30}T_{64} = 2^{30}\frac{T_1}{S_{64}} = \frac{2^{31}}{S_{64}}s$. If we have a linear scaleup, it will take about $2^{31}s$, which is about 68 years to brute force the 56 bit key. Extrapolating our scaleup analysis, S_{64} will be a much smaller number, so the time to brute force a 56 bit key on a 64 core machine will be very large.