Assignment 2: Measuring Parallel Performance

Detian Deng

March 5, 2015

Hardware

Amazon EC2 c4.4xlarge, High-CPU Extra Large Instance, with 62 EC2 Compute Units on 16 virtual cores of 8 physical CPUs.

Part 1: Parallel Coin Flipping

Analysis

1. Scaleup and speedup

- Produce charts that show the scaleup and speedup of your program. Please Use line charts with correctly labelled axes and ticks. Do not use bar graphs or scatter plots.

Fig.1 Coin Flip Speedup

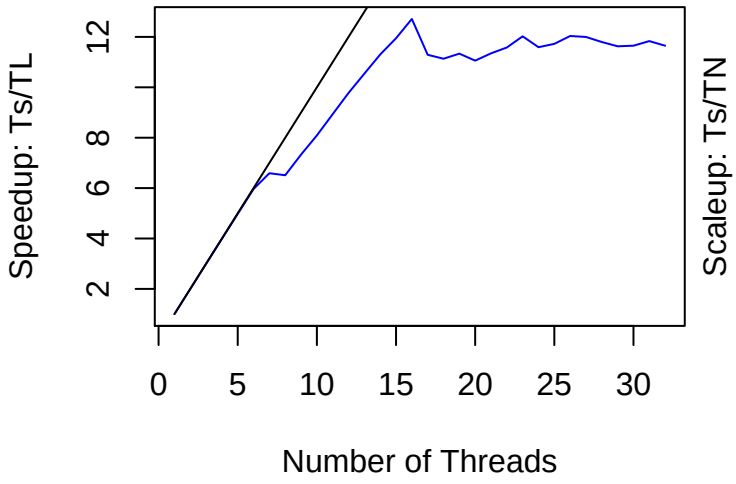
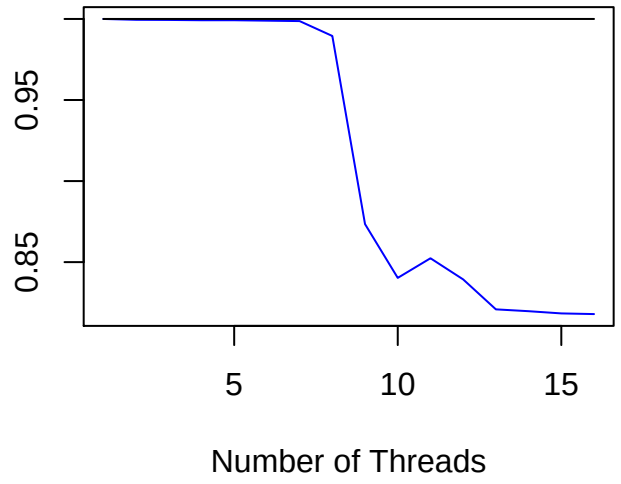


Fig.2 Coin Flip Scaleup



- Algorithm (true) speedup/scaleup measures the scaling performance of the algorithm as a function of processing elements. In this case, from 1..8. Characterize the algorithmic speedup/scaleup. If it is sub-linear, describe the potential sources of loss.

From 1 to 8 threads, the algorithmic speedup and scaleup are both sub-linear. The major reason is the startup cost and synchronization in multiple threading. When the number of threads increases, more startup time is required to initialize all of them. Also, at the end of each thread, each thread needs to synchronize its individually calculated number of heads with the total number of heads. The more threads, the more time it will take to synchronize.

- Why does the speedup not continue to increase past the number of cores? Does it degrade? Why?

From Fig.1, we can see that the speedup stops increasing after number of threads larger than 16. With 16 virtual cores, the maximum number of threads that can run at the same time is 16, then the additional threads will not get exucuted until one of the ongoing threads finishes. That means these additional threads are acting like serial code and cannot increase the speedup. In fact, it will degrade because for example, using 17 threads means one of the 16 cores need to run two threads each with 1/17 of the task sequentially and it costs more time than each core running one thread with 1/16 of the task. Moreover, since more threads causes additional time to initialize and synchronize their results, the speedup will degrade after threads number gets larger than 16.

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

- Describe your experiment. Why does it measure startup?

For the `CoinFlip.java` code, the startup cost mainly involves creating the array of Thread objects and initialize all threads, thus I seperated the `.start()` part from the initialization part and measured the startup cost by the following chunk of code.

```
long starttime = System.nanoTime();
// Array to hold references to thread objects
Thread[] threads = new Thread[numthreads];

// create and start specified thread objects of class SynchronizedWorks
for ( int i=0; i<numthreads; i++ )
{
    threads[i] = new Thread ( new CoinStartUp(i, numIter, numthreads) );
}
long headuptime = (System.nanoTime() - starttime)/1000;
```

- Estimate startup cost. Justify your answer.

For each number of threads from 1 to 16, the program runs 5 times. The average startup time in **microseconds** are summarized below. As we can see, the startup cost increases as the number of threads increases.

	1	2	3	4	5	6	7	8
Average Startup Time	82.00	88.20	94.60	102.80	108.60	114.80	120.60	126.80

	9	10	11	12	13	14	15	16
Average Startup Time	131.60	140.60	148.00	153.20	160.80	166.60	173.00	178.80

- 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.)

Let the number of threads be k, the total running time using k threads be $T(k)$, the startup costs (serial) fraction be s. Based on Amdahl's law, we have

$$T(k) = T(1)(s + \frac{1-s}{k})$$

$$\frac{T(k)}{T(1)} - \frac{1}{k} = (1 - \frac{1}{k})s$$

Therefore we can estimate s by a simple linear regression with no intercept using the speedup time data with number of threads from 1 to 8.

The estimated s is 0.0087. Therefore, based on the Amdahl's law, the speedup using 100 threads on independent physical cores, $S(100)$ is 53.6533, $S(500)$ is 93.3884, and $S(1000)$ is 102.9156.

	Estimate	Std. Error	t value	Pr(>|t|)
x	0.0087	0.0045	1.92	0.0958

Table 1: Fraction of Startup Costs (Serial)

Part 2: Brute Force a DES Key

Analysis for Part 2

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

For speedup, the key size is fixed at 20, and the number of threads ranges from 1 to 16.
For scaleup, the key size ranges from 20 to 24, while the corresponding number of threads are 1,2,4,8,16.

- Produce charts and interpret/describe the results. Please Use line charts with correctly labelled axes and ticks. Is the speedup linear?

Speedup chart is shown in figure 3, and scaleup chart is shown in figure 4. Both speedup and scale up are sub-linear. For the number of threads from 1 to 5, the speedup curve grows fairly fast, but after that, the trend is that the more threads we used, the less incremental speedup we got. The scaleup decreases relateively fast comparing to the coin flip program.

Fig.3 DES Key Speedup

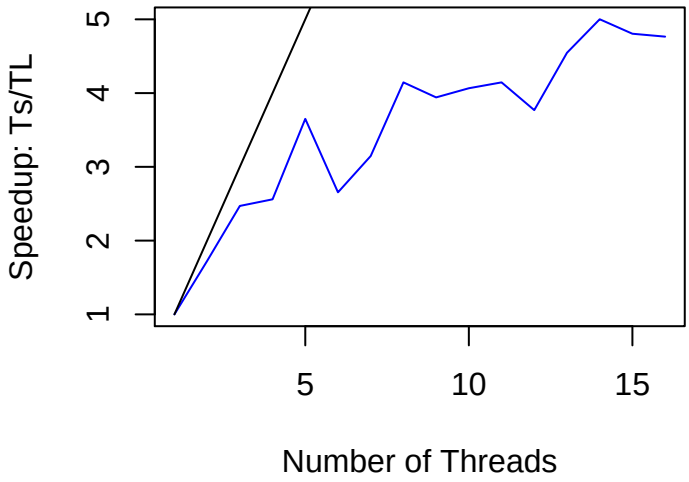
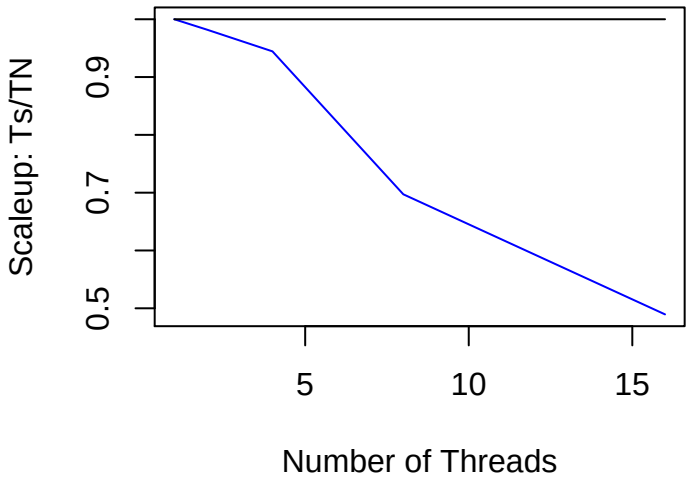


Fig.4 DES Key Scaleup



- Why do you think that your scaleup/speedup are less than linear? What are the causes for the loss of parallel efficiency?

Both scaleup and speedup are sub-linear because of because of the following reasons:

1) The fraction of the serial code in the program which involves creating the key and encrypting the message is non-trivial and increases as the key size grows. Thus by Amdahl's law, the speedup is sub-linear, and the scaleup degrades faster than it would do if the fraction was constant.

2) The startup cost, which involves initializing all threads and creating a new `SealedDES` object in eath thread , increases as the number of threads grows.

3) The Amazon EC2 c4.4xlarge instance has 8 physical CPUs, which means for the number of threads 9 to 16, hyperthreading is involved. But hyperthreading is not equivalent to creating a thread on an independent processor, it is a way to let two threads make better use of the register space on a single physical processor. Thus its performance gain is lower than having a thread running on an independent CPU.

- 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.

Based on our knowledge, we can fit a linear model to predict the running time when the key bit is 56 and the machine has 64 physical cores.

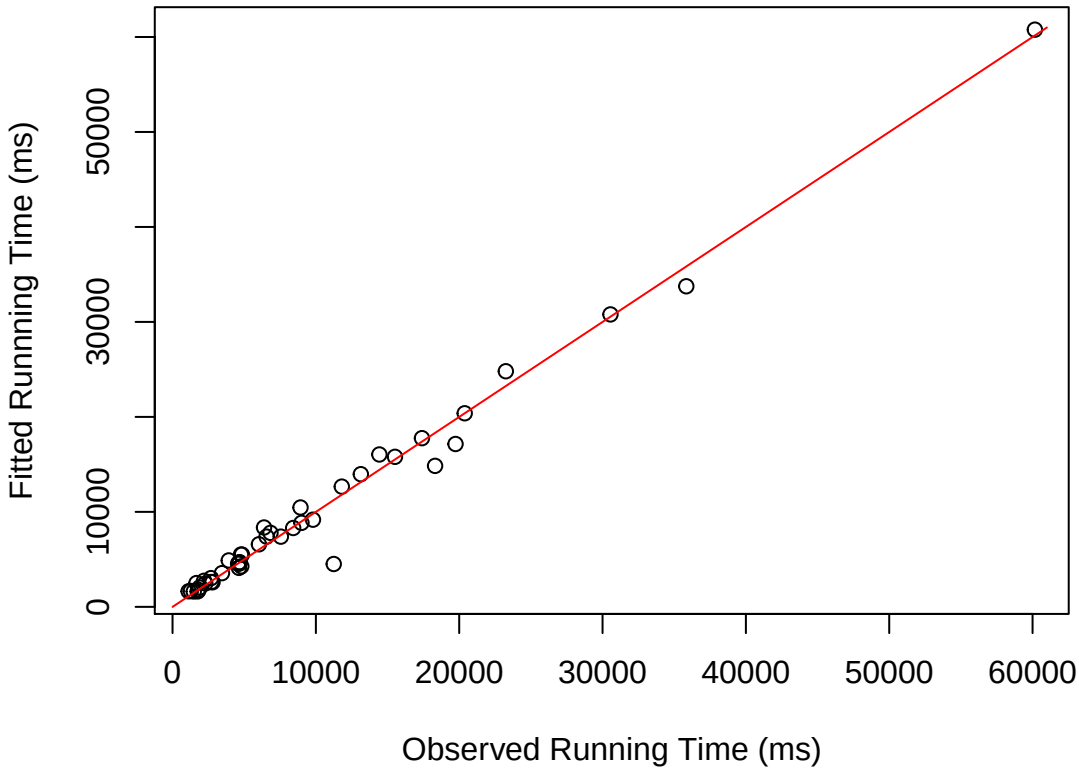
Let L be the key length in decimal,$l = \frac{L}{2^20}$, k be the number of cores, T be the total running time, $S = S_0 + kS_1 + lS_2$ be the serial fraction running time, $t = \frac{1}{k}(t_0 + lt_1)$ be the parallel fraction running time. Then the linear model can be written as below:

$$T(k,l) = S_0 + kS_1 + lS_2 + t_0\frac{1}{k} + t_1\frac{l}{k}$$

Data from speedup and scaleup experiments up to 8 threads were used. In addition, more data were generated for each combination between thread number 1 to 8 and key bits 21 to 24.

The fitted model parameters are summarized below:

	Estimate	Std. Error	t value	Pr(>|t|)
S1	91.1536	73.0873	1.25	0.2196
S2	403.7990	64.9628	6.22	0.0000
t1	3344.1541	145.8749	22.92	0.0000
t0	714.6167	783.6431	0.91	0.3673



To evaluate the out of sample prediction, running time for 8 threads 27 bit size was also generated, whose average is 95420 miliseconds. And the model predicted value is 1.0601 $\times$ 105 miliseconds, which is fairly close to the observed value.

Therefore the extrapolated running time is 3.134 $\times$ 1010 seconds, which is 993.7726 years.