# Project 2 Analysis Part

*Bohao Tang*

*March 3, 2018*

## Introduction:

You can fine the original data used here in `coindata.txt` in https://github.com/bhtang127/cs420-parallel/tree/master/project2 . I ran the code handed in 20 times and use the median of them to do the plot. The standard deviation for each data is small for about 1-2 ms.

## Part 1: Parallel Coin Flipping

### 1. Factors Against Parallelism

Below we have provided the source for a common implementation of the `next` method of the `Random` object. The next method is always used to generate random objects, regardless of type (float, int, bool, etc). The `nextInt` method, for example, calls `next` with a bit value of 32.

```
protected synchronized int next(int bits)
{
    seed = (seed * 0x5DEECE66DL + 0xBL) & ((1L << 48) - 1);
    return (int) (seed >>> (48 - bits));
}
```

Given the source code above, imagine that you created a single Random object shared across each of your threads.

#### a. What type of speedup (e.g. linear, sub-linear, constant) would you expect to see?

**Answer:** Constant speedup.

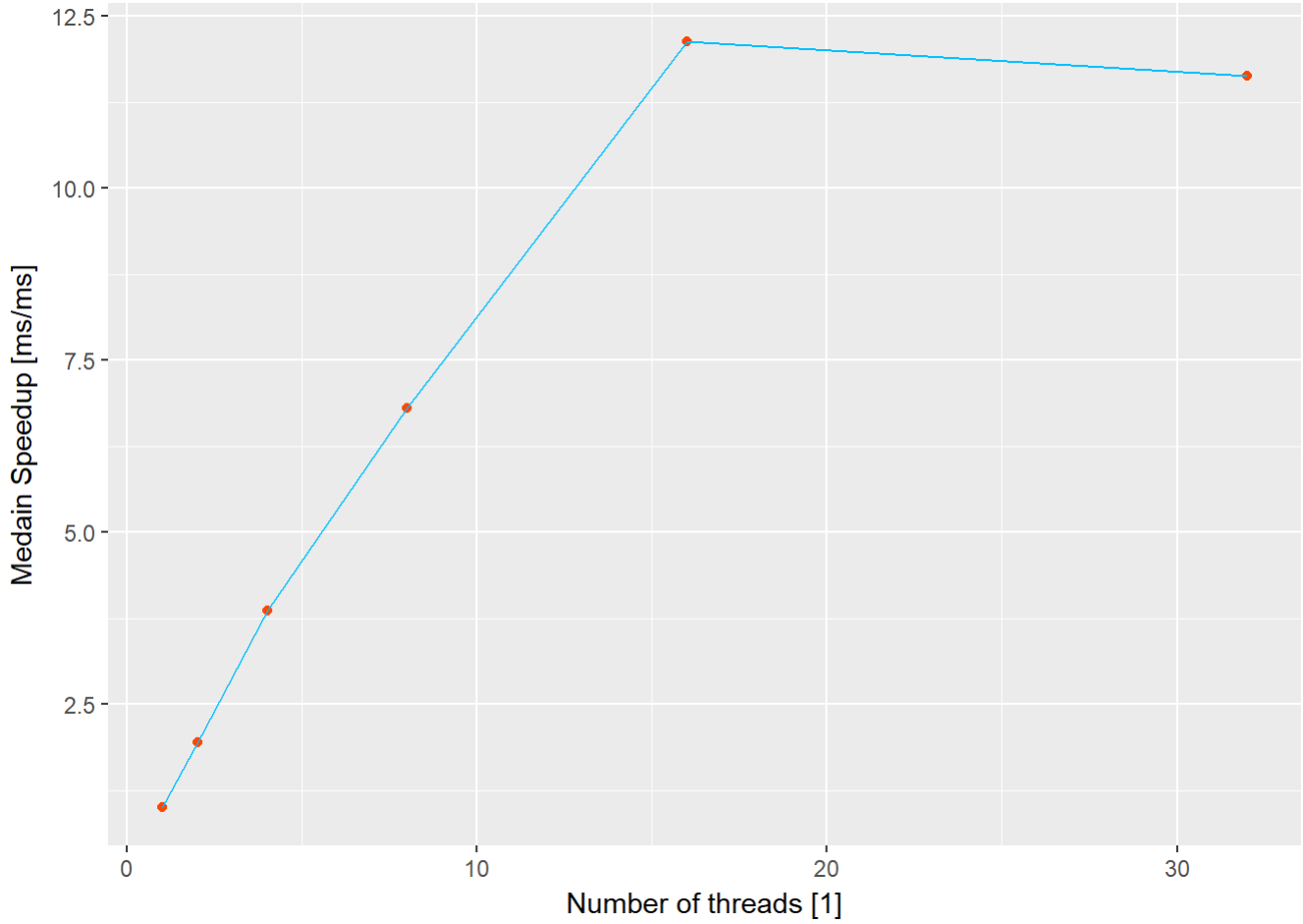
#### b. What factor against parallelism does this demonstrate?

**Answer:** This is due to the interference effect. When you share the random generator with all threads, the `next` method should be an atomic step, therefore when one thread is getting its random number (therefore updating `seed`), the system should add lock to `seed` meanwhile blocking all other threads from getting their random numbers. So whatever the thread number is, the generator is actually generating numbers one by one and distributing them to each thread. Since all threads can do their job only after they get the next random number, there won't be any speedup.

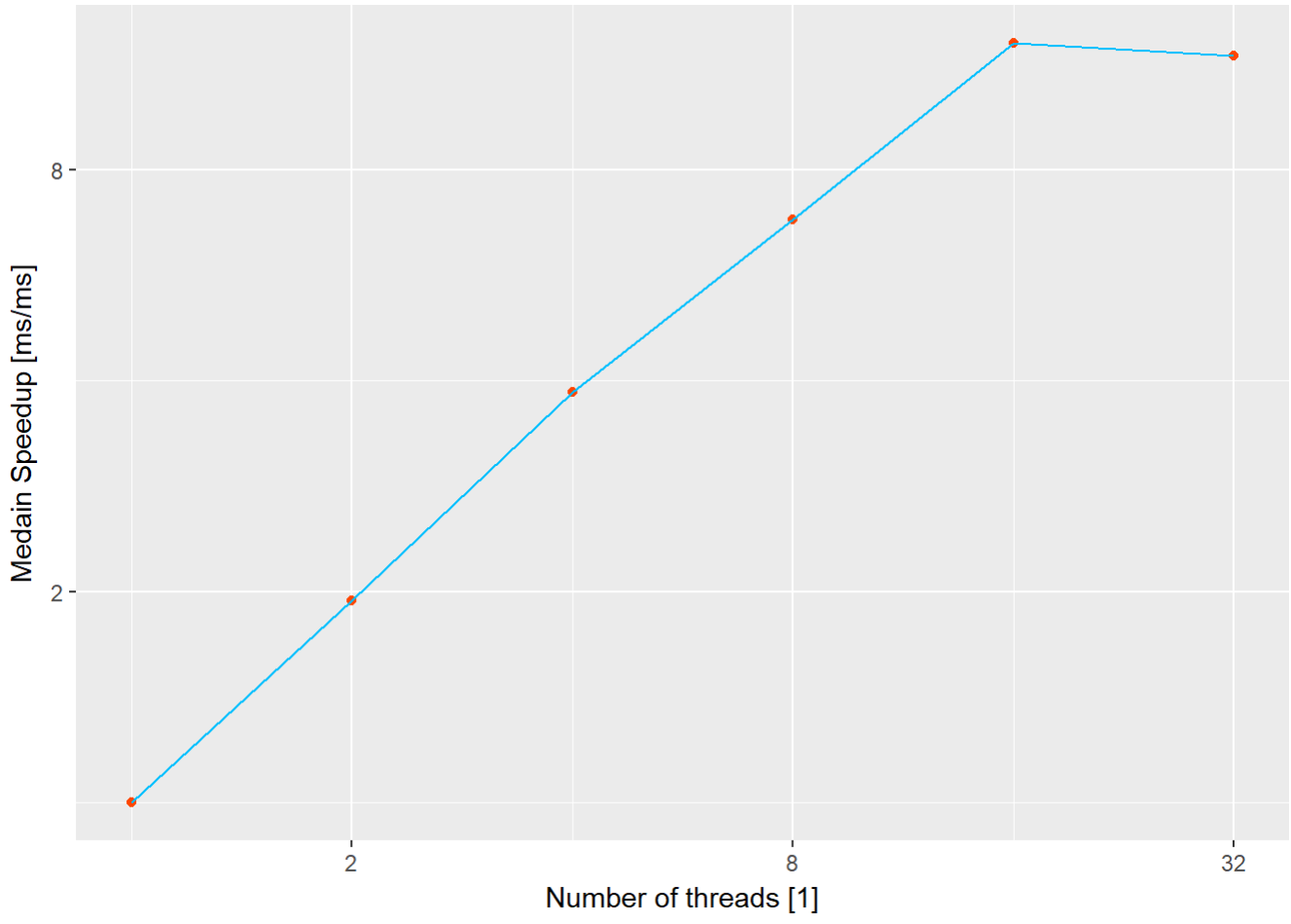
### 2. Scaling concepts (please clearly label all axes and include units and scale)

#### a. Produce (i) a speedup plot and (ii) characterize it in terms of linearity.

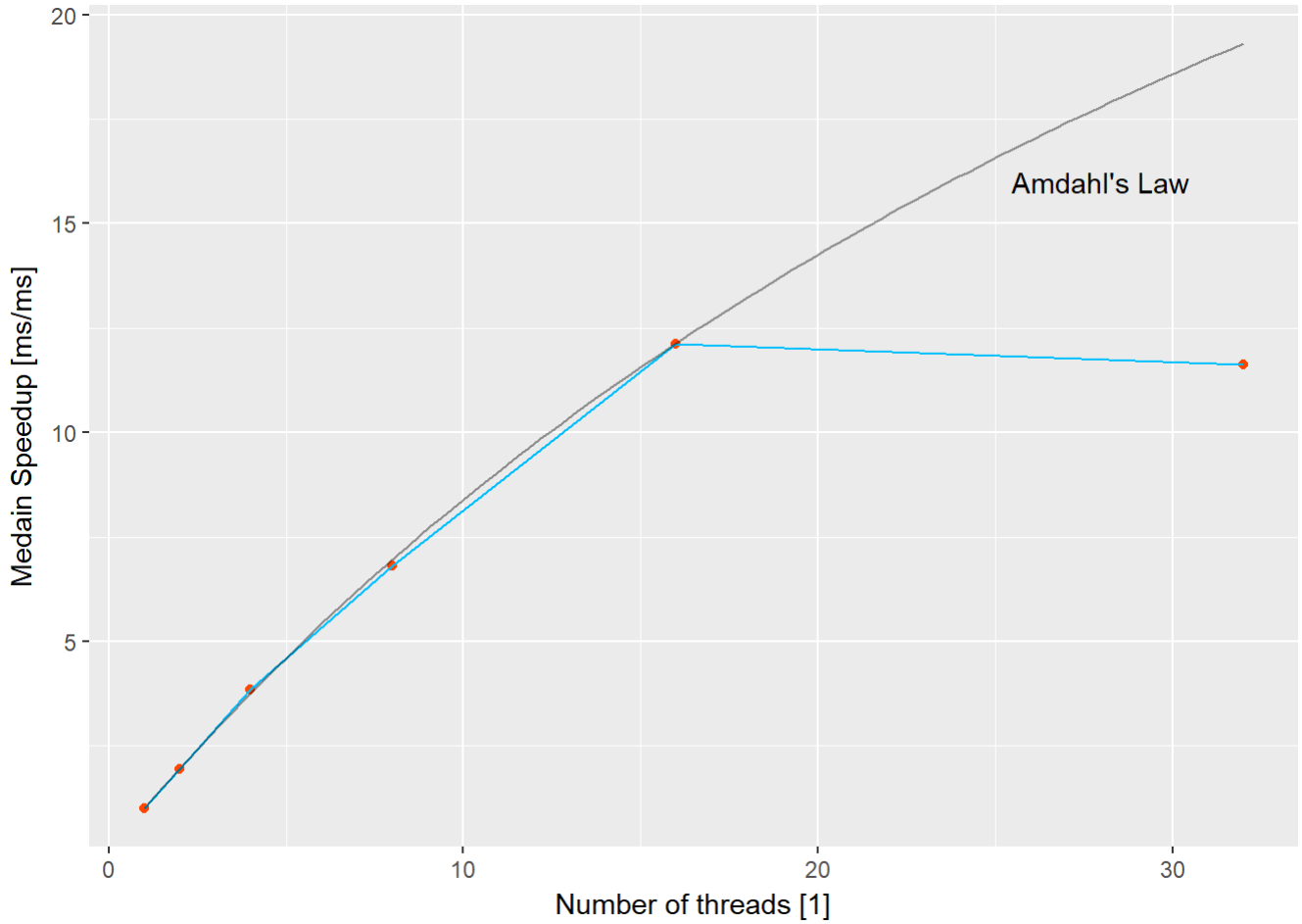
**Answer:** (i). The speedup plot is:




And here is a log-log version of plot:



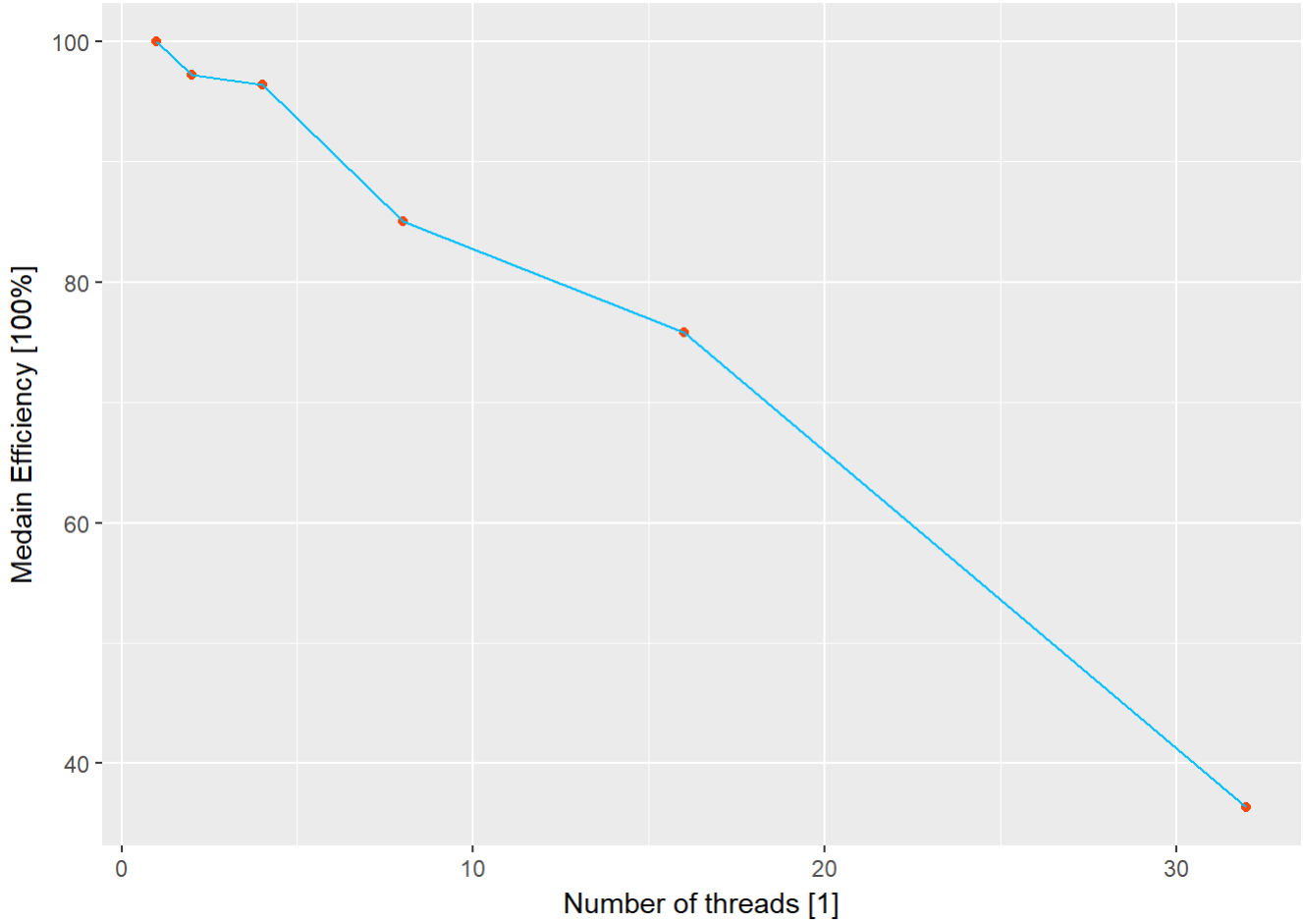
(ii). We can see that the speedup curve seems very linear (a little sublinear) until thread_num = 16, and then suddenly tail off with no further increasement. This is because the server only have 16 vcpu therefore can not fully parallelize the program with 32 threads. Here is an estimated plot (light grey part) of curve for Amdahl's law (before thread_num 32), the estimated `p` is 0.9788:




Therefore we can see the speedup curve is rather sublinear and follows Amdahl's law well before threads 32.

### b. Produce a (i) parallel efficiency plot and (ii) characterize it in terms of linearity.

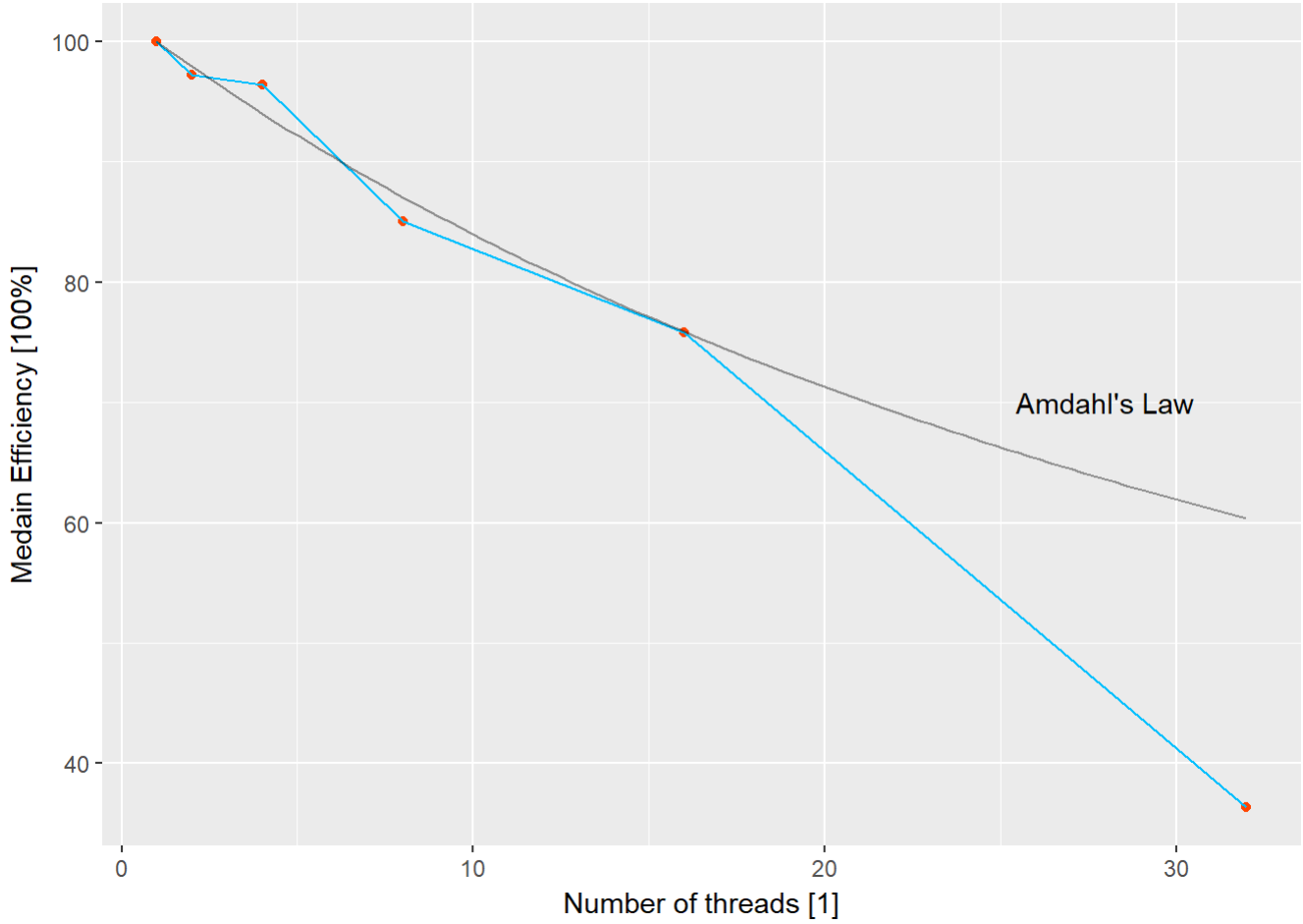
**Answer:** (i). The parallel efficiency plot is:




(ii). The parallel efficiency curve seems like a linear decrease. We have shown in previous question that the data follows Amdahl's law with `p=0.9788` well when `num_threads < 32`. Since `p` is near 1, `num_threads * (1-p) / p` is small. Therefore we have that parallel efficiency:

$$\text{Efficiency} = \frac{100\%}{1-p+\frac{p}{n}}/n = \frac{100\%}{p}\frac{1}{1+n\frac{1-p}{p}} \simeq \frac{100\%}{p}(1-n\frac{1-p}{p})$$

Where `n` is the `num_threads`. Therefore the data will look like a linear decrease at least when `num_threads` < 32. Here is a plot of the data together with the estimated efficiency by Amdahl's law:




# Part 2: Brute Force a DES Key

See the code