

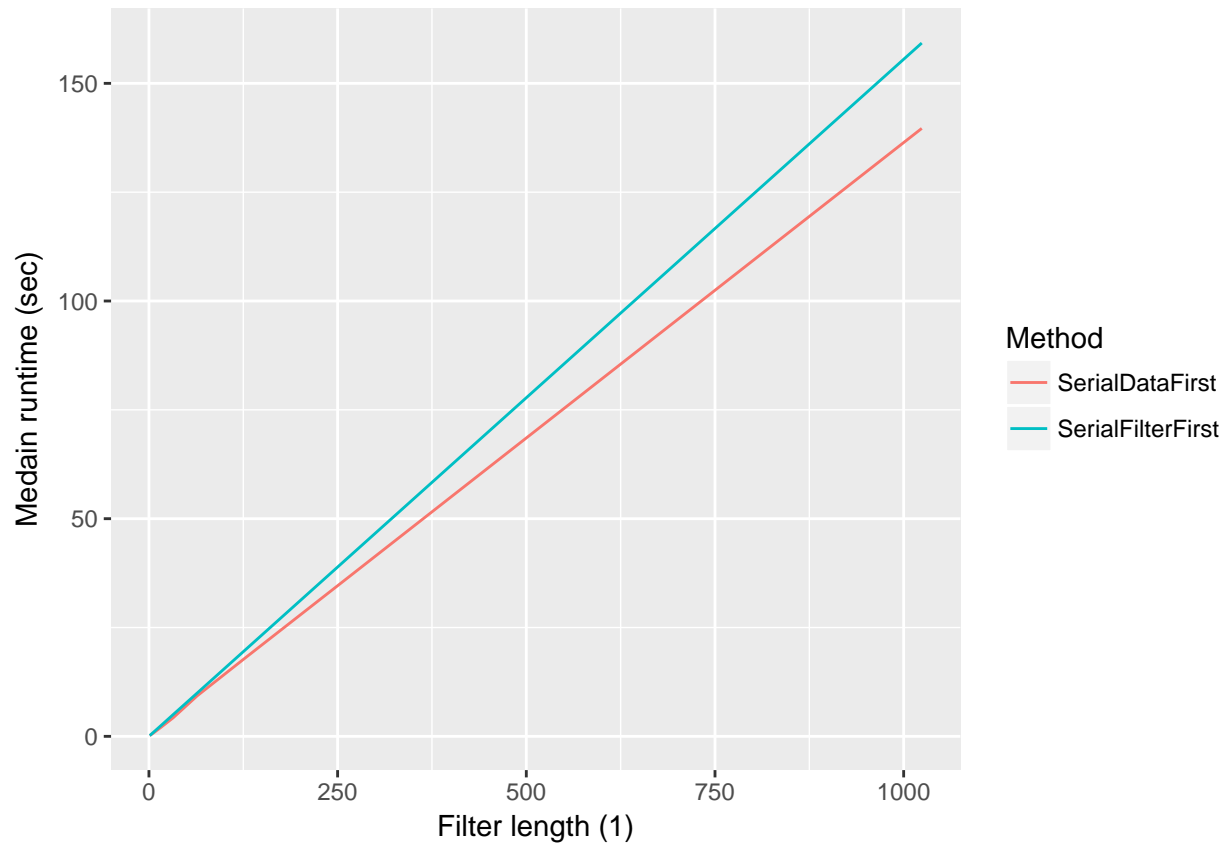
# CS420 Project1

*Bohao Tang*

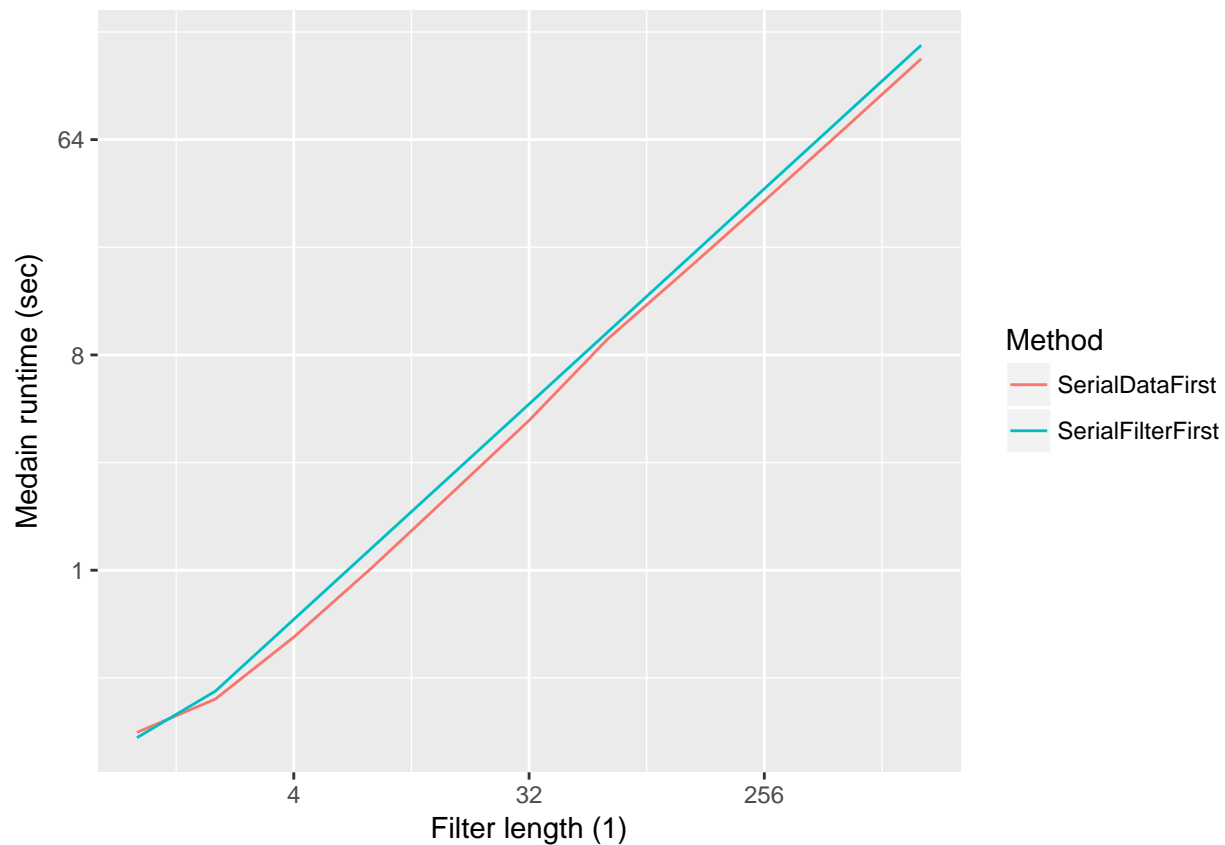
*February 20, 2018*

## Loop Efficiency

a. Here is the plot of runtime vs filter size:

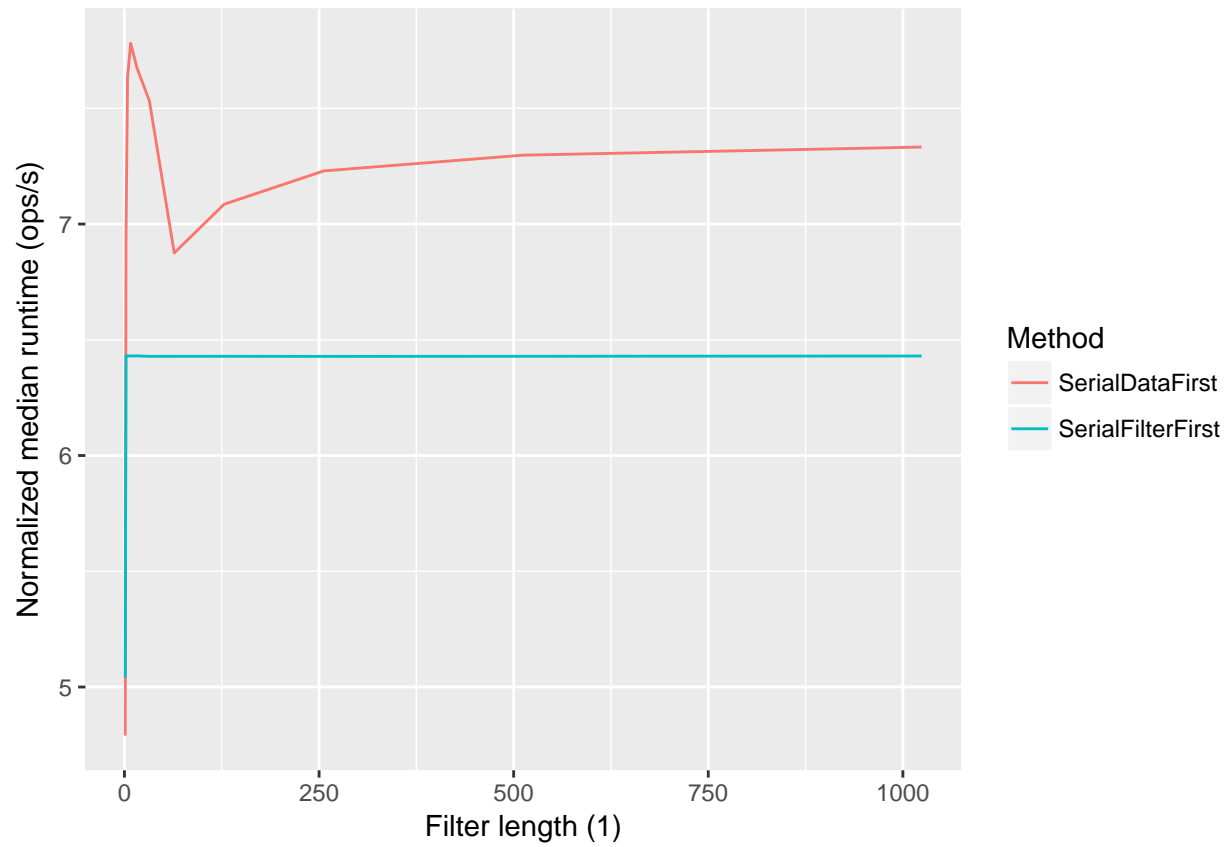


And here a log-log version plot of it:

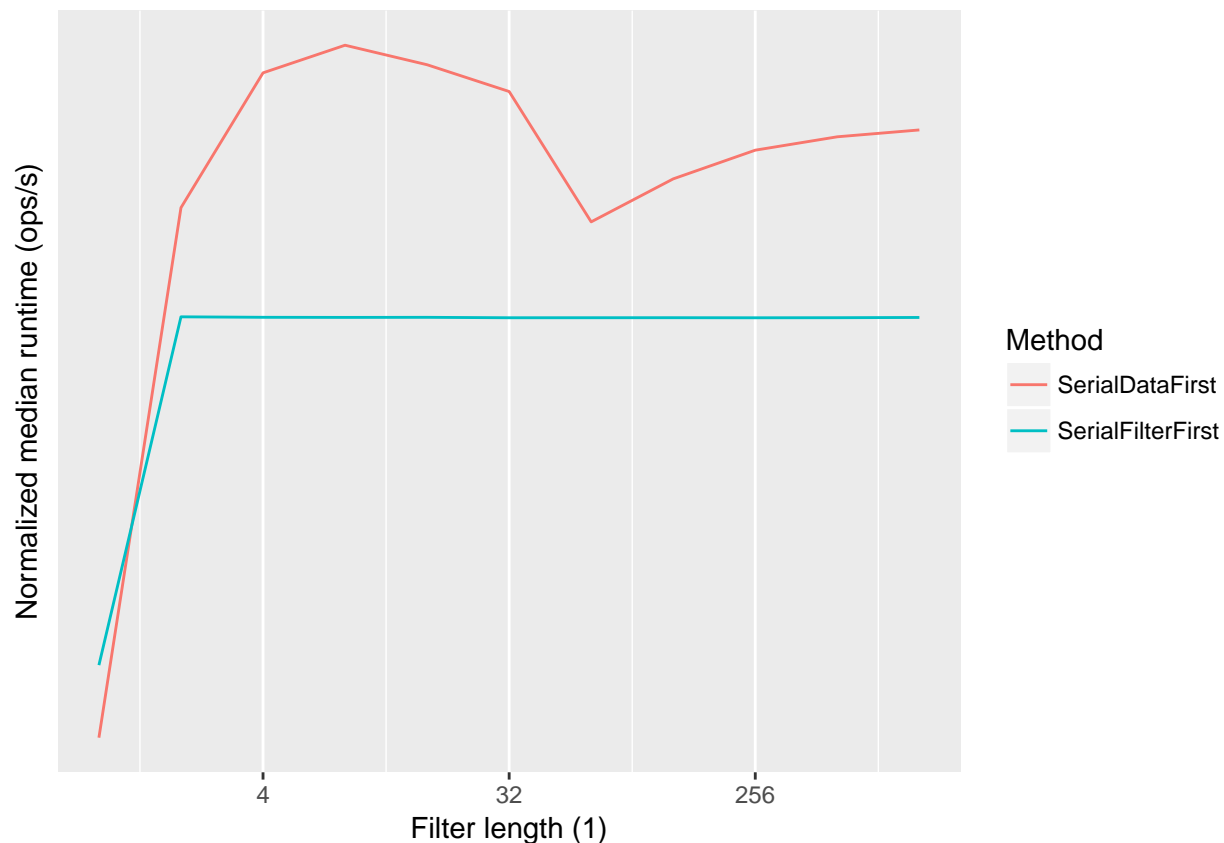


We can see that when filter length bigger than 1, function `serialDataFirst` costs less time.

- b. In the implemented functions, total number of operations is approximately `filter_length * data_length`. Since `data_length` doesn't change, we normalized the runtime by `filter_length / runtime`. Then here is the plot of normalized runtime vs filter length.



And here is a log-log version plot:

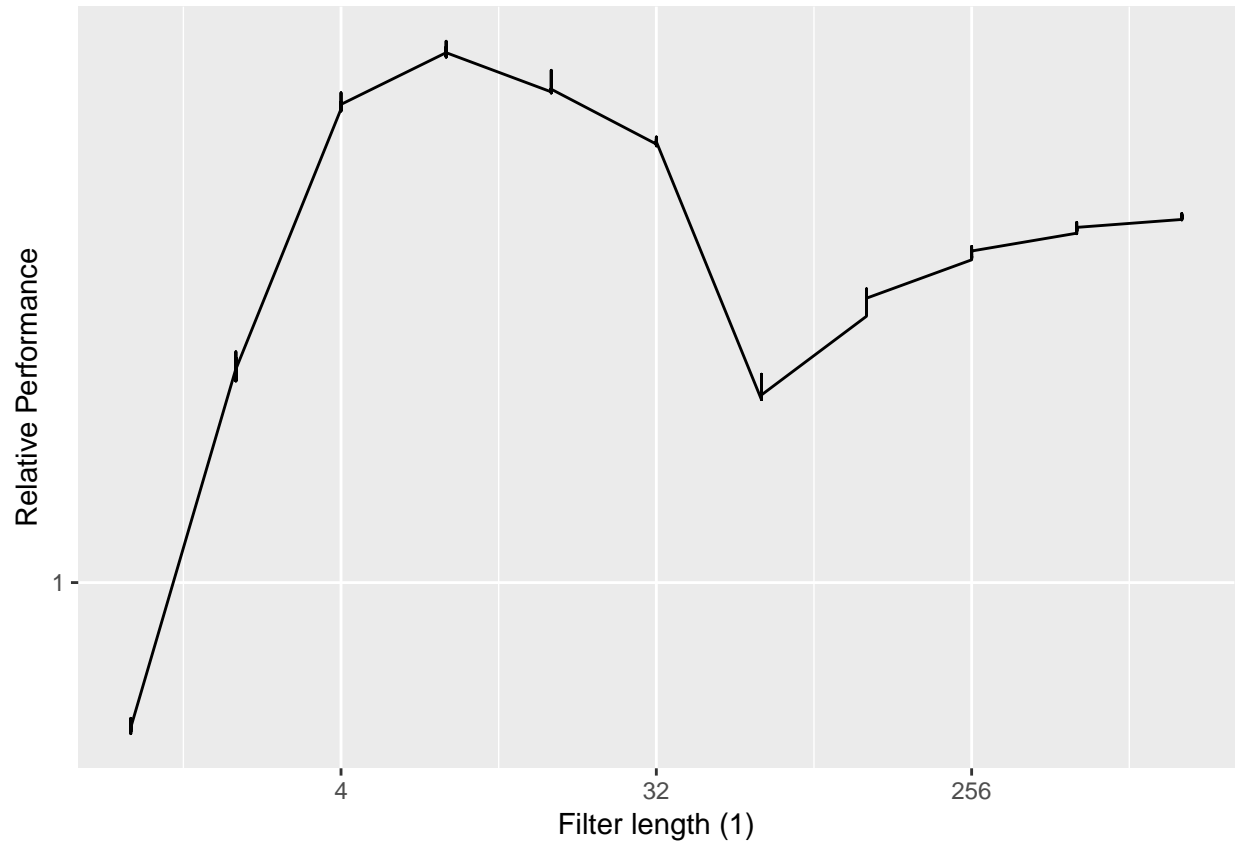


We can see here that when `filter length > 1`, function `serialDataFirst` can run more operations per second.

- c. It is more efficient to have data in the outer loop. Because here `data length` is very big `512*512*256` and `filter length` is relative tiny at most 1024. So it is likely that when you fix a filter element and scan the data, data can not be totally read into memory(cache) meanwhile the filter can when you fix a data point and scan the filter. That will cause a different i/o speed and then putting data outside will be faster.

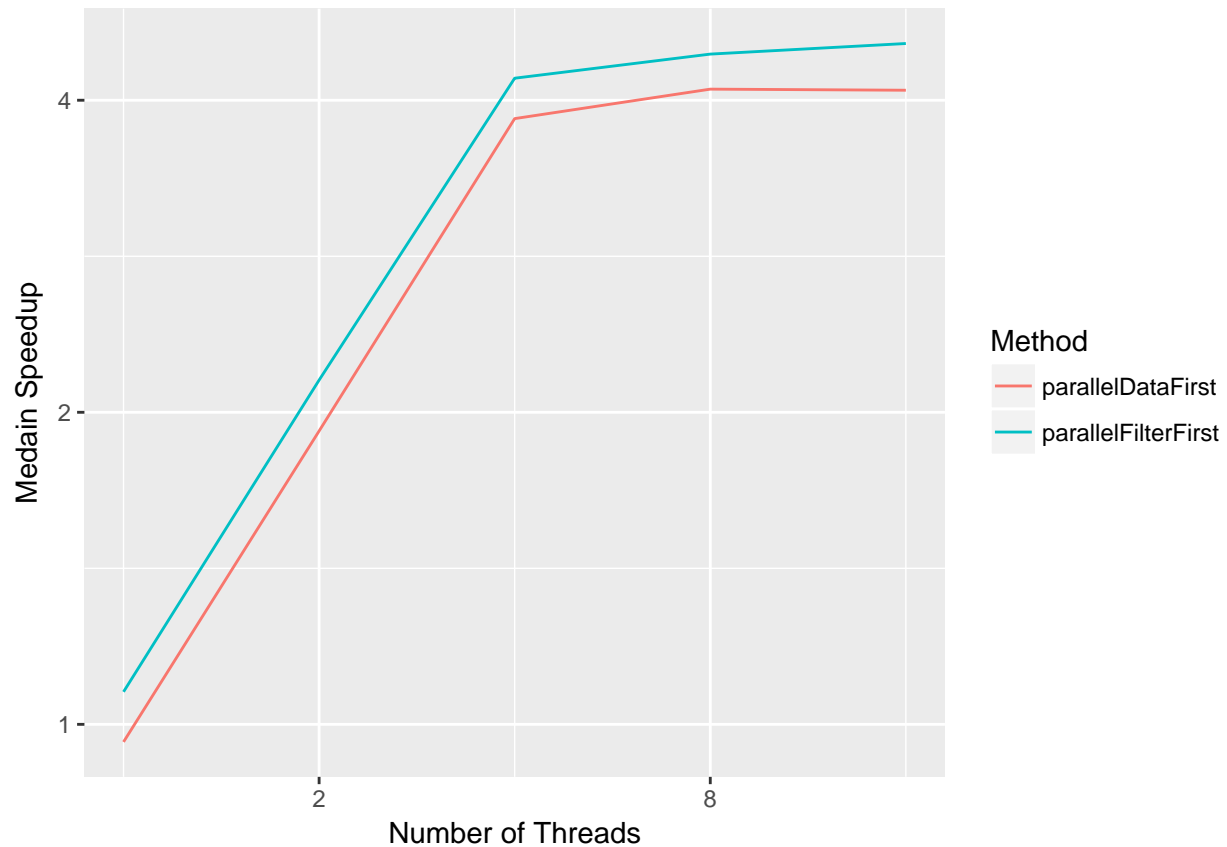
But when filter size is equal to 1, `serialFilterFirst` is faster. This is because of the loop overhead. When you put data outside you will keep jumping statement between inner and outer loop, and this will slow down the program.

- d. We divide time of `serialFilterFirst` by that of `serialDataFirst` to get the relative performance, and the log-log plot is like below.



## Loop Parallelism

1.
  - a. See the code
  - b. See the code
2.
  - a. Here is the plot for speed up.



b. The speedup increase approximately linear before threads 4, slightly increase between 4 and 8, and tailed off after 8. This is strong scaling because the speedup curve tailed off and also the total problem size never change. This program is run on c5.2xlarge which has 8 cores, so ideally speedup curve will increase linearly up to 8 threads because it has 8 independent computing centers and then curve will be sharply tailed off.

3.

a. For the absolute time, **parallelDataFirst** is still faster because the problem mentioned above (Loop Efficiency 1.c.) still holds. But as the plot above, **parallelFilterFirst** have a more speedup, maybe this is because filter size is small and when you divide them you will reduce the jump of statement more than dividing larger data. Or because that filter is small, so dividing them will make them able to hold in higher cache and therefore higher speed.

b. fff

c. ggg

## An Optimized Version

a. ppp

b. qqg