Assignment 1 Solutions

Shaojie Chen

February 24, 2014

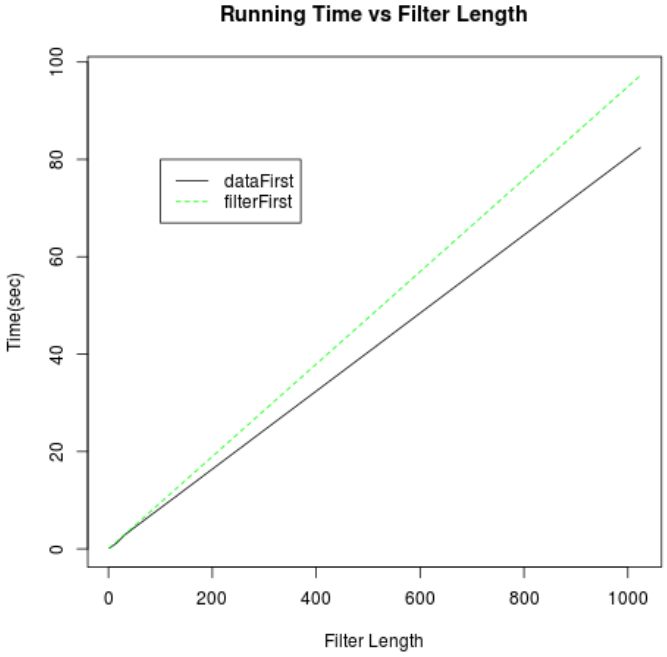
1 Loop Efficiency

Hardware Specifics

I am running all the simulations on AWS a c3.2xlarge instance with 8 CPUs of 2793.364 MHz and 14 GB memory. All simulation results are averages over 20 simulations. No outliers are dropped out.

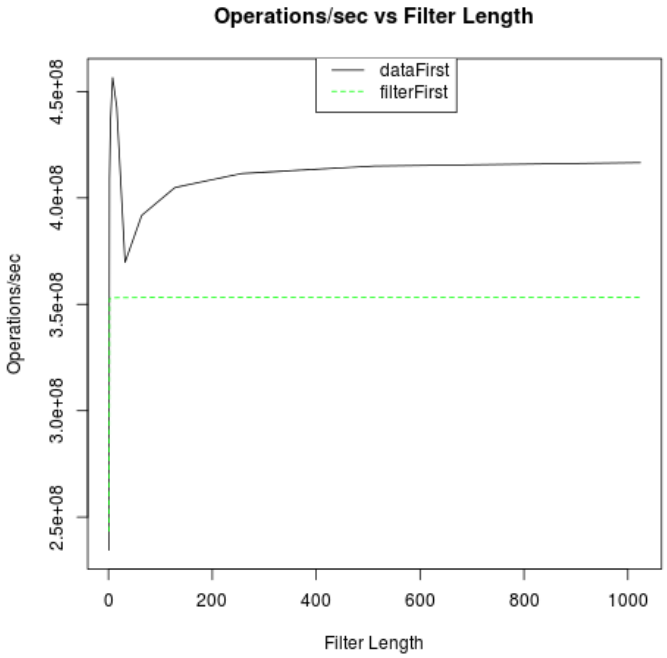
a.

The serial running time versus filter length plot is shown in figure 1. We could observe that the running time is linearly increasing with filter length. Also it takes less time if we put data vector in the outter loop.



b.

The number of operations per second versus filter length plot is shown in figure 2. Here I used the product of data length and filter length as an approximate of the number of total operations. I have two observations. First, the operations per second is relatiely low for shorter filter length, because the overheads take up a significant part of total running time. When the filter length increase, the overheads is only a very small portion of total running time and the operations per second is approaching a constant level. Second, the operations per second is higher if we put the data vector in the outer loop, because the cache is better utilized.

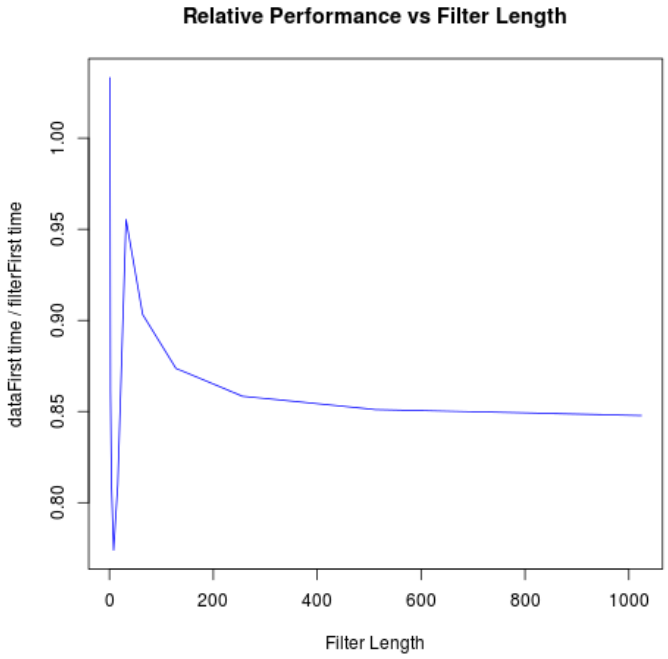


c.

It is more efficient to have the data in the outer loop. This is true because of having the data in the outer loop will take advantage of cache sharing. To be specific, if we put data in the outer loop, then in each inner loop, the input_array[x] is a constant and only requires one load for all the inner loops. But if we put filter in the outerloop, input_array[x] is updated for each inner loop, while filter_list[y] keeps constant. Since the data vector is longer than the filter vector, there are more savings by putting the data in the outer loop.

d.

The relative performance (data first running time / filter first running time) plot is shown in figure 3. For a shorter filter length, the data first method has no big advantage (for filter length of 1, the filter first method is even slightly better). As the filter length increase, the relative performance approaches a constant around 0.85.

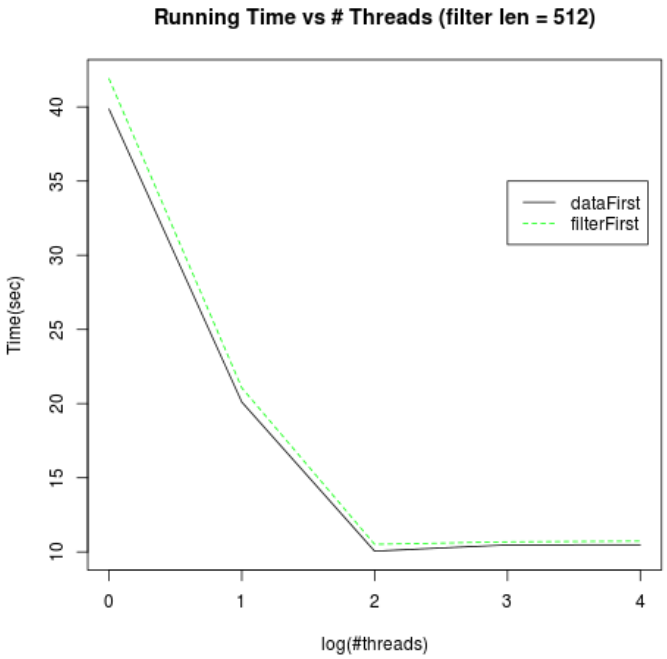


2 Loop Parallelism

speedup and scale up.

a.

The speedup plot (here I am using running time, we could definitely use a speed up equivalently) is shown in figure 4.

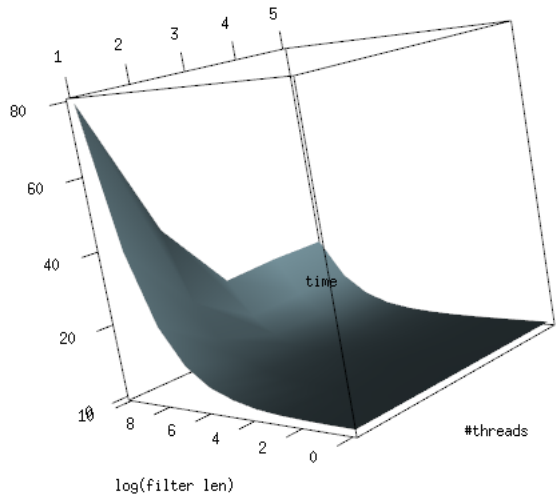


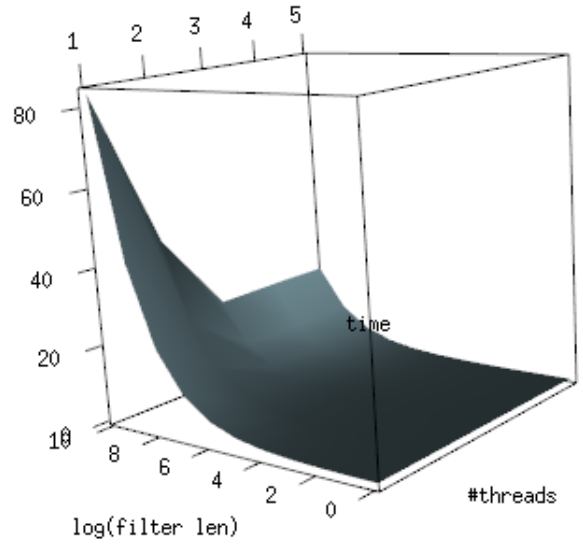
b.

From the previous speedup plot, we could see the running time first decreases and then approaches some constant with the increase of threads numbers. An speed up of about 4 times is achieved when the number of threads is greater than 4. In this parallel simulation, data first method is still performing slightly better than filter first method, but not as significant as in the serial one. The plot tails off around 2 to 3 (correspondingly 4 to 8 threads). I am running the simulation on a AWS c3.2xlarge instance with 8 CPUs. When the number of threads is around the number of cores, best performance could be achieved because further increasing number of threads will lead to waiting of some threads.

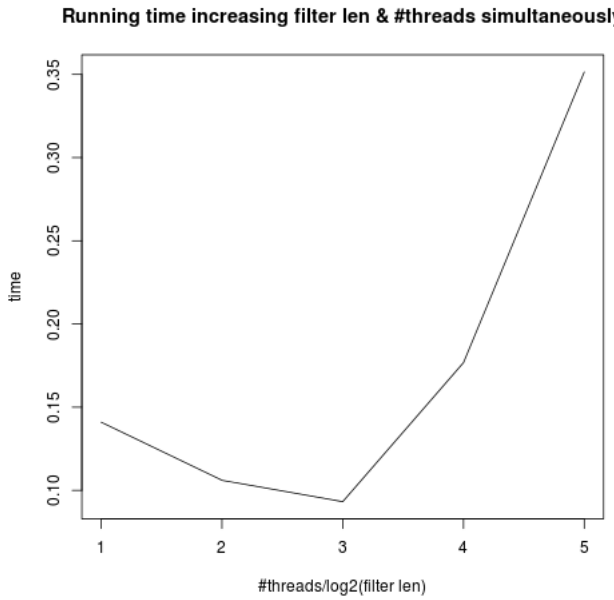
c.

Here I am first giving a 3D plot of running time versus number of threads and filter length, as shown in figure 5. The two plots represent data first method and filter first method respectively.


1
2
3
4
5
80
60
40
20
10
8
6
4
2
0
time
#threads
log(filter len)


1
2
3
4
5
80
60
40
20
10
8
6
4
2
0
time
#threads
log(filter len)

To look at the scale up from another view of point, I increase the number of threads as 1,2,4,..,16, while increasing the filter length as 64,128,256,512 and 1024 simultaneously. The plot is give in figure 6.



d.

Parallel Performance.

a.

The data first parallel method is still a little faster. The reason is similar to the serial case, the data first methods still better utilized the cache within each thread. The advantage over the filter first method is not as significant because the time saving is not accumulated but spreaded over multiple threading, that's why the two lines in the speedup plot is getting closer as the number of threads increase.

b.

According to Amdahl's law, if the parallel portion of code is p, number of threads is s, then the speed up

$$n = \frac{1}{(1-p) + \frac{p}{s}}$$

rearranging the terms, we got

$$\frac{1}{n} - 1 = (\frac{1}{s} - 1)p$$

This is a simple linear regression with no intercept. Plug in the data from our simulation, we estimate a p = 0.8728 from data first method and a p = 0.8791 from filter first method.

c.