Calculating Stock Covariance and Variance in Parallel

Matt Butts

June 2, 2014

Contents

1 Introduction

1.1 Big Data

Big data sets can model nearly everything in the physical and theoretical realm. With increased access to data comes the need to model and describe everything around us calling for new ways to decipher accurate meaning from this data without a complete loss of efficiency. We have access to this data, now we need a means of analyzing it. A solid approach is parallel computing. This allows you to separate processes to different processors to be computed simultaneously and then be reassembled. This can solve a lot of complexity issues, especially those that arise from excessive iterations. Obviously, another concern is the language being used because different programming languages operate with different efficiencies. I'm not interested in finding the most efficient language for this project, just developing methods that can handle large data sets efficiently. However, if I had more time and a better understanding of the C programming language I would implement Cython functionality to compute the intermediate calculations in parallel.[3]

1.2 Finance

Big data is huge in finance. There are a huge amount of values that go into calculating particular relationships between stocks. Some of these relationships that are more notable are called moments and co-moments. They can be translated to more commonly known terms like mean, variance, skewness, kurtosis, etc. Some of the more commonly analyzed moments and co-moments are the first and second moments/co-moments. I will denote a moment with $M_{a,X}$ and a co-moment with $C_a(X, Y)$. The mean of a variable is the first moment, denoted

$$M_{1,X} = \bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

The second moment is known as the variance and denoted,

$$M_{2,X} = \sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n}$$

. The first co-moment is the mean of the data and the second comoment is the covariance of the two variables, which I will talk about in more detail in the next section. [5]

1.3 Approaching the Calculation of Covariance

I want to use a two-pass approach to calculate the covariance. I prefer this method because it is less prone to residual errors. A one-pass algorithm to calculate covariance tends to fall victim to unnaceptable numerical instability. This can cause the calculated covariance to be 0 when it really isn't equal to

0. This tends to be more of an issue on big data sets. A single-pass algorithm does the calculation $COV(X,Y) = \sum_{i=1}^{n} \frac{(x_i - \bar{x})(y_i - \bar{y})}{n}$. n represents the number of elements in the list. The difference in the two-pass algorithm is only that the

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

and

$$\bar{y} = \sum_{i=1}^{n} \frac{y_i}{n}$$

values are calculated first. Then they are referenced throughout the calculation of

$$COV(X,Y) = \sum_{i=1}^{n} \frac{(x_i - \bar{x})(y_i - \bar{y})}{n}$$

. The same concepts of one-pass and two-pass algorithms apply almost identically for variance calculations. Also, by making this algorithm a two-pass algorithm and separately saving the x and y means of the set the calculation can be parallelized. It is parallelized by separating the entire list into chunks, making covariance and variance calculations on the individual chunks, then reassembling them in a pairwise fashion. [2]

2 My Algorithm

2.1 Additional Packages

I just used the package "pandas" to read my .csv files because it does so cleanly and efficiently.

2.2 File Input/Chunk Processing

A user runs the "intializer module" function with 3 parameters. The first is a file path to the ".csv" file with all of the relevant stock returns for the two stocks to be analyzed. The second and third parameters let the user specify whether or not to print and whether or not to save the result to a file. Then the file is passed to a function that processes it into equal or close-to-equal sized chunks and they are saved into a list that holds all of the data.

2.3 Chunk Calculation

I divide the list into 4 sub-lists, as close to the same length as possible. This part of my algorithm uses the methods to calculate variance and covariance that are described in section 1.2 and 1.3. The method is a two-algorithm for each. This means that first the mean is computed and then the variance or covariance calculation is made. This helps ensure the result will be numerically

stable and be more accurate. This is also essential in the chunk assembly step because the data means are part of the equation that combines the chunks back to the original data set. This is the part of the program that is computationally expensive. So, this is the part of the program where I implemented the parallel decorator to break up the complicated processes to be ran in parallel. I do a straightforward, two-pass, calculation of covariance. I do the calculation in parallel by using the parallel decorator. This is accomplished by inputting a list of inputs to my parallel decorated function. Then, it takes this list and computes each list entry in parallel. [4]

2.4 Chunk Assembly

I compile the four separate pieces of data into one with a recursive function that does a pairwise combination of the dictionary data, with two total passes before the recursion is broken. The data set size and other linear operators are updated with simple formulas [5] Most importantly, data set means and sample sizes are maintained so that the calculation can occur at this step of the process to reassemble all of the pieces. The assembly process is pairwise and first the mean must be calculated, then coefficients $deltaX$, $deltaY$, and $gamma$, and finally the variances and covariance.

The mean is assembled with the pairwise equation

$$\bar{X} = \frac{(n_A * \bar{X}_A) + (n_B * \bar{X}_B)}{n_A + n_B}$$

The variance is updated with the pairwise equation

$$Var_X = \frac{(n_A * Var_A) + (n_A * Var_A) + (n_A(\bar{X}_A - \bar{X})^2) + (n_B(\bar{X}_B - \bar{X})^2)}{n_A + n_B}$$

[2]

First, the following coefficients are calculated:

$$delta_X = \bar{X}_B - \bar{X}_A$$

$$delta_Y = \bar{Y}_B - \bar{Y}_A$$

$$gamma = \frac{(n_B^2 * n_A) + (n_B * n_A^2)}{(n_A + n_B)^2}$$

The covariance is updated with the pairwise equation

$$Cov_{X,Y} = \frac{(Cov_A * n_A) + (Cov_B * n_B) + (delta_X * delta_Y * gamma)}{n_A + n_B}$$

[1] After the covariance and variances are calculated over the entire list a function runs some calculations to obtain the standard deviations and correlation for the data as follows:

$$Corr_{X,Y} = \frac{Cov_{X,Y}}{StdDevX * StdDevY}$$

$$StdDevX = \sqrt{Var_X}$$

$$StdDevY = \sqrt{Var_Y}$$

2.5 Output

The output will either be saved in a file or printed on the screen, depending on the parameters passed by the user. The format is a string with an introductory message and a dictionary. The dictionary has the following key-value pairs:

covXY covariance of the data sets, X and Y

corrXY correlation of the data sets, X and Y

deltaX delta coefficient of the data set X

deltaY delta coefficient of the data set Y

meanX mean/average of the data set X

meanY mean/average of the data set Y

varX variance of the data set X

varY variance of the data set Y

nX data points in the data set X

nY data points in the data set Y

std devX standard deviation of the data set X

std devY standard deviation of the data set Y

3 Conclusion

A speed comparison between my parallel algorithm and a standard two-pass algorithmic approach is featured in the below table. It's clear that my algorithm isn't as efficient for small data sets, but quickly surpasses the normal algorithm when using data sets of around 5,000 - 10,000 points. Also, note that all of the speeds in the table are "per loop" to help ensure that the calculations are unbiased. Also, data points is the number of points per set so in reality the number of points being processed in the function is equal to double of that shown; that is, if you count the points in each set then the number is double that in the table below.

3.1 Table of Time Comparisons

DATA POINTS	MY ALGORITHM'S SPEED	COMMON APPROACH SPEED	RESULTS
1,000	60.9 ms	19.3 ms	Slower
5,000	89.9 ms	88.6 ms	Almost equal
50,000	232 ms	897 ms	3.86x faster
100,000	393 ms	1.76 s	4.48x faster
500,000	1.72 s	9.02 s	5.24x faster
1,000,000	3.37 s	17.8 s	5.28x faster

References

[1] Janine Bennett. Numerically stable, single-pass, parallel statistics algorithms. http://www.janinebennett.org/index_files/ParallelStatisticsAlgorithms.pdf.

[2] Tony Chan. Updating formulae and a pairwise algorithm for computing sample variances. ftp://reports.stanford.edu/pub/cstr/reports/cs/tr/79/773/CS-TR-79-773.pdf.

[3] Michael Stonebraker. What does big data mean? http://cacm.acm.org/blogs/blog-cacm/156102-what-does-big-data-mean-part-2/fulltext.

[4] Sage Development Team. Sage documentation. http://www.sagemath.org/pdf/en/reference/parallel/parallel.pdf.

[5] Wikipedia. Algorithms to calculate variance and covariance. http://en.wikipedia.org/wiki/Algorithms_for_calculating_variance.