Computing Covariance and Variance of Stock Returns in Parallel

Matt Butts

University of Washington

June 2, 2014

Common approaches to practical applications of calculating statistical moments are slow because of the computational complexity that comes with huge data sets. This is a problem in finance when computing stock relationships because computing these moments requires iterations over those return data sets.

The first moment is the mean of the list.

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

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

The second moment is the variance of the list.

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

Standard Deviation (units of the data)

$$StdDevX = \sqrt{\sum_{i=1}^{n} \frac{(x_i - \bar{x})^2}{n}}$$

The second co-moment is the covariance of the two lists.

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

The correlation (standardized units)

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

•My project is a Sage progam that can compute correlation and standard deviation of large data sets of stock returns quickly by calculating covariance and variances in parallel.
•These large data sets can be partitioned into blocks and expensive calculations made in forked processes then reassembled to represent the entire data set.

Simple Breakdown of Algorithm:

1. Input/List Processing
2. Calculate Chunk
3. Chunk Assembly
4. Output

File Input/Chunk Processing:

- My program reads a CSV file with column data representing stock returns for two stocks
- User can specify to save to a .txt file and/or print results
- The lists are divided into blocks to computed in parallel

Calculate Chunk:

•Performs computations to convert return lists into dictionaries of data.

•Parallel Decorator runs list of partitions of original list in parallel forked processes

•Running forked processes in this step makes this algorithm efficient for larger data sets.

Chunk Assembly:

•Pairwise assembly algorithm that assembles the four chunks, now represented with dictionaries, into one that represents entire data set.

Output:

- My program can save/print the output as a dictionary of the moment/co-moment data
- It calculates correlation and standard deviation because these are considered standardized measures

I ran the same data sets on my parallel program and a program that runs a similar algorithm, not using parallel processes.
Data Points are per set and speed is calculated per loop using "timeit"

Data Points	My Algorithm	Common Approach	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

If I had more time and knowledge to make this program faster:

- Implement cython in the computationally heavy functions
- Somehow optimize how processes are run in parallel
- Optimize the number of list divisions based on original data list size and idea of pairwise assembly

Questions?