KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Stock price correlation calculation

Project

Benjamin Rübenkamp
R0793577
MEIsw
Master industrieel ingenieur Elektronica-ICT
Academiejaar 2023-2024

** There are no code examples in this text, please look in github repository under project: github link **

Introduction:

This report discusses a CUDA implementation designed to calculate the correlation between the price movements of different stocks. The implementation uses the GPU to speed up the calculations using reduction and shared memory. The program reads historical stock price data (OHLC data as in Open,High,Low,Close data) from CSV files, calculates the price returns and then the correlation between two stocks.

Code structure and functionality:

The program consists of several components:

Calculation of returns: The calculateReturns kernel calculates the price return for each record in the data. The price return is defined by the following formula:

$$\frac{close - open}{open}$$

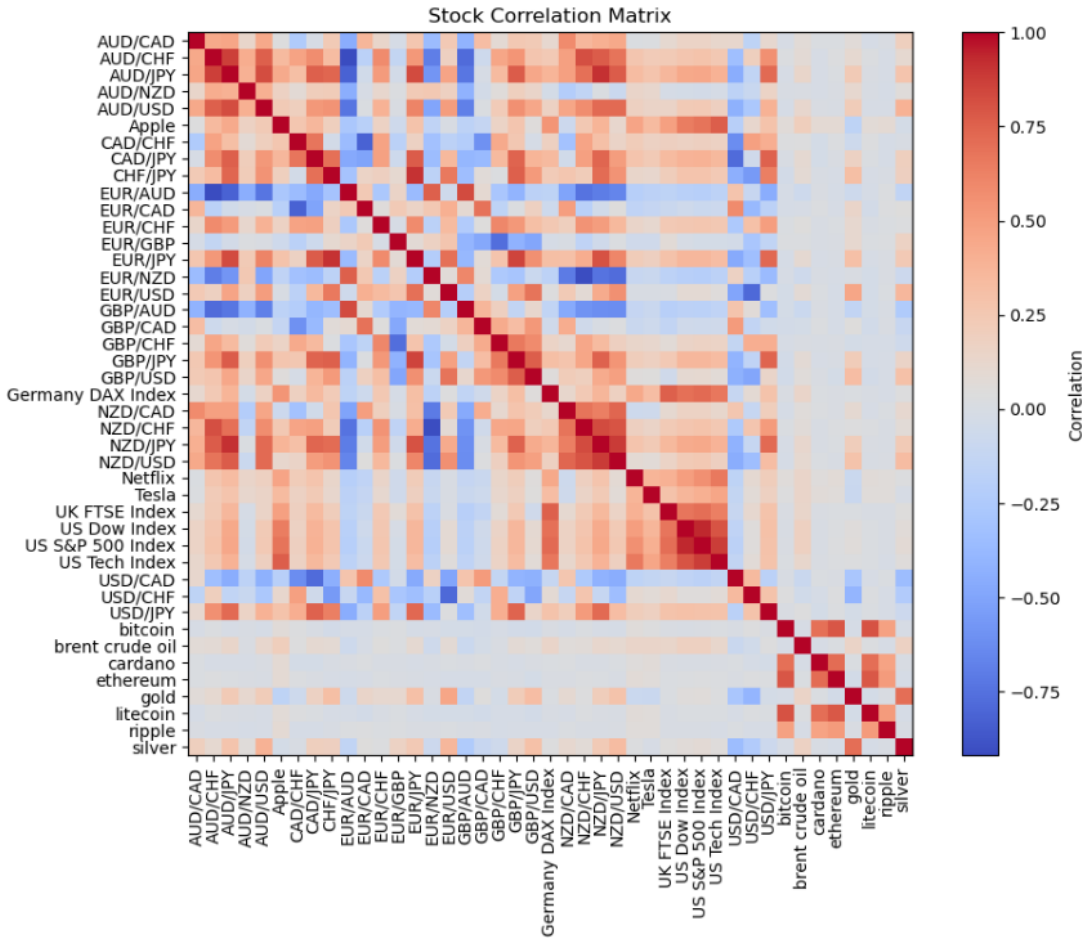
Calculating averages: The kernel calculateMeans calculates the average values for the price returns of two stocks. This is done in three steps: first the sum of all elements in a block are calculated, then the global sum is determined, after which the total sum is divided by the number of returns to get the mean value.

Calculation of correlation: The calculateCorrelation kernel calculates the correlation between the price returns of two stocks. The correlation factor r is calculated using the following formula:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

Figuur 1: Correlation factor formula

Result



Figuur 2: Correlation matrix

Applications of the output:

The output of this program is the correlation coefficient between the price returns of two stocks. This correlation coefficient can be used for various financial analyses, including:

Portfolio diversification: A low correlation between two stocks suggests that their price movements are independent of each other. This can be used to construct a portfolio that is less sensitive to fluctuations in individual stocks.

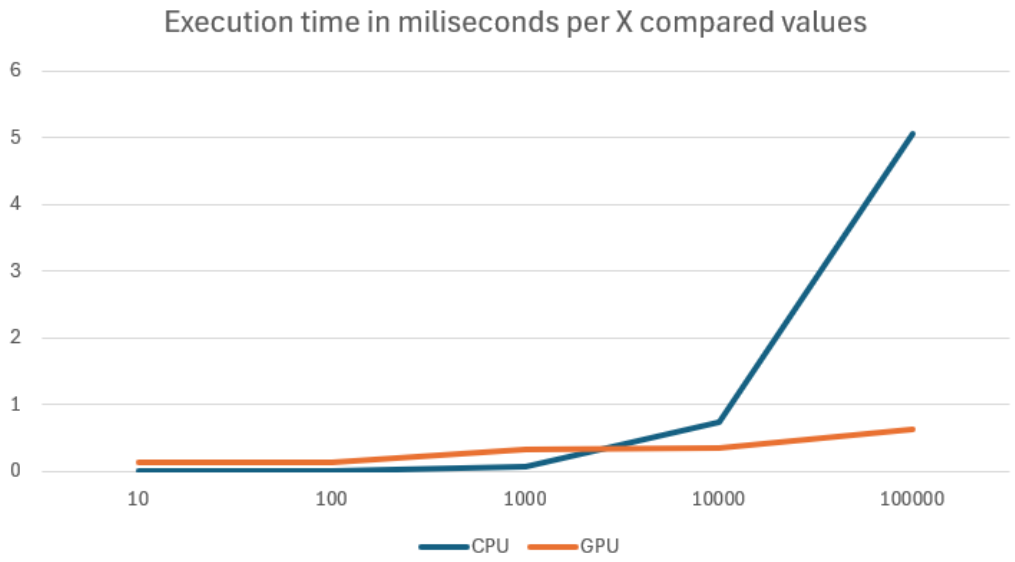
Pairs Trading: Stocks with high correlation often exhibit similar price movements. This can be used in pairs trading strategies, where an investor goes long in one stock and short in another, with the expectation that the historical correlation will continue.

Risk management: Correlation analysis helps assess a portfolio's risk. Stocks with low correlation offer diversification benefits, which can reduce the overall risk of the portfolio.

Market forecasting: Changes in the correlation between two stocks can indicate changes in market conditions or industry trends. This information can be used to make trading decisions and predict future market trends.

Speed comparison GPU vs CPU

Although each calculation takes only a few milliseconds, comparing 40+ different price series with any other price series, with X number of price points each time takes quite a while. However, performing the calculation is not the most time-consuming activity. The most time consuming activity is all the data processing that happens before the calculations. For example, each iteration you have to compare all timestamps to find and calculate the correlation just the common datapoints, put the common ones into new lists, and then take the first X number of those. (And for the GPU allocating memory and starting threads.) The following graph only shows the comparison between the execution times of the calculations. When calculating the total execution time for 1 calculation of the correlation, results don't make sense in the way that sometimes the iteration with 1k elements has a longer duration than the iteration with 10 elements. It is clear that when there are many datapoints, the GPU is a lot faster using reduction and shared memory.



Figuur 3: Execution times

Possible improvements

For calculating total values (such as total sum of all elements when calculating the average), the different blocks by reduction each calculate their own sum, which will end up in shared memory at index 0. To calculate the global sum, each first thread of a block writes that first value (total sum of within block at index 0) away to global memory. The global first thread then merges all these written away values from global memory afterwards. This process could be improved by using an atomic add so that each first thread within a block can add it's sum to the global sum. Unfortunately the atomicAdd instruction only works for Floats and not for doubles. You can multiply a double value by 1000 before the atomic add and then divide it by 1000 when reading the total value to make up for the loss in accuracy but I have not been able to get this working in time.

Another improvement would be to calculate returns for both price series in the same kernel, which would also speed up the calculation by minimizing thread spawning overhead.

I also noticed that above a limit of about 10k data points the correlation values are no longer correct and not the same every time. At first I thought that the shared data size limit was exceeded, but the size of shared data is adjusted according to the block size, and therefore independent of the total number of elements. In my opinion it cannot be the fault of the global memory size either, since for example 50k elements with block size 1024 only 50 double values need to be in global memory.

Conclusion

Correlation calculation between stock prices using CUDA provides valuable insights for investors and financial professionals, allowing them to make more informed decisions when managing their portfolios and executing trading strategies. Executing the calculations in parallel on the GPU is a time saving option.