ECE1724S Programming Massively Parallel Graphics Processors

Final Project Report

John Pazzelli - #990981069
Tijmen Tieleman - #994557742

John Pazzelli - #990981069
Tijmen Tieleman - #994557742

Introduction

Background

The Foreign Exchange (Forex) market offers traders the ability to trade the world's major currencies in a fast-paced, commission-free marketplace. The market organizes currencies into pairs, such that one may be purchased by selling another. For example, the 'EUR/USD' symbol corresponds to the Euro/US dollar currency pair whose value is the current exchange rate when converting Euros to US dollars.

Many Forex traders decide to buy/sell specific currency pairs by analyzing charts that show how the exchange rate changes in real-time. This analysis generally involves one or more numerical 'indicators' that are graphed on top of the exchange rates that attempt to show the current market trend and predict the future market direction. For example, in the chart below, the real-time rate data (the jagged curve) is overlaid with a simple moving average (SMA) indicator curve that displays the average of the most recent 50 values at each point:

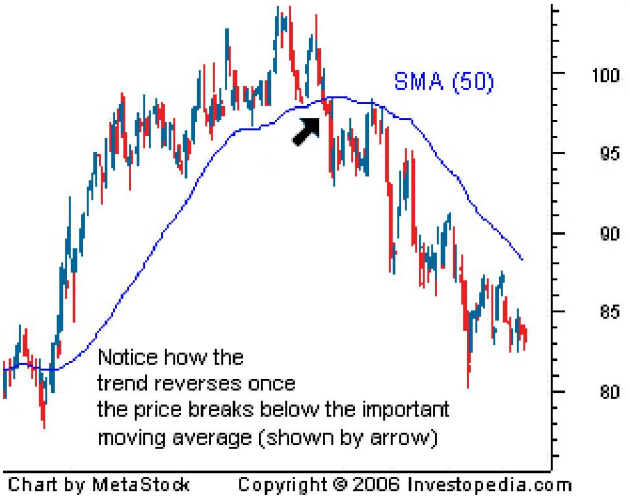

Fig. 1 – Simple Moving Average Indicator (Janssen, Langager, & Murphy, 2009)

Similarly, many other indicators have been developed that attempt to show trends in the data (for example, the Exponential Moving Average indicator applies exponentially decaying weight factors to each sample, giving more weight to newer samples and less weight to older ones).

Chart indicators such as the moving average curve in figure 1 produce buy and sell signals (called 'breakout points') at specific points on the chart. In figure 1, a breakout point occurs when the moving average curve crosses above the exchange rate curve, indicating a potential reversal in the previous uptrend – i.e. a 'sell' signal, indicated by the black arrow). Although there are many commercially-available charting software packages that will calculate and display these indicators, few (if any) attempt to determine the optimal parameters to use when calculating a given indicator. For example, in the case

of moving averages, the users themselves must specify how many samples the moving average should be based on, producing a different curve and ultimately different buy/sell signals along that curve. The optimal curve is the one that most closely tracks the data fluctuations during the analysis period (i.e. produces buy/sell signals that are the most accurate) and therefore produces the highest profit.

Project Focus and Rationale

The focus of this project was therefore to use the GPU to attempt to determine the optimal parameters to use when generating a few different technical analysis indicators. The rationale for a GPU-based approach is that if the optimizations were part of a real-time, electronic trading platform, the market conditions may change in the time required to have the CPU determine the optimal parameters, effectively making the results irrelevant. For a small number of simple indicators, the CPU alone may be 'fast enough', but if optimizing across a large number of more complex indicators, the need for high performance computations increases.

Project Results

This project found that use of the GPU can considerably reduce the time required to perform indicator optimizations, with more complex indicators containing multiple dimensions of optimization (such as MACD) benefiting the most from the GPU approach. The speed improvement for the EMA-S and EMA-OC indicators that contain only one dimension of optimization was 8x, while the EMA-D indicator (two dimensions of optimization) was 75x, and the MACD indicator (three dimensions of optimization) was 133x. Optimization of the CCI indicator (3 dimensions of optimization) was also performed, though these dimensions were found to be independent of each other and could be calculated independently, resulting in a very fast CPU implementation and therefore eliminating the need to create a corresponding GPU implementation.

Algorithm Implementation Details

This project attempted to implement parameter optimization on the following technical indicators over a set of N input data samples:

Table 1 – Technical Analysis Indicator Descriptions

Code	**Indicator Name** **(Breakout Point Identification)**	**Optimization Parameters** **(Range of Optimization)**
EMA-S	Exponential Moving Average – Single (Close price crosses over MA)	# of samples in MA (2 to N)
EMA-OC	Exponential Moving Average – Single (Open and close price cross over MA)	# of samples in MA (2 to N)
EMA-D	Exponential Moving Average – Single (Short and long curves cross)	Long curve: # of samples in MA, i = (3 to N) Short curve: (2 to i) for each long curve produced
MACD	Moving Average Convergence/Divergence (MACD line crosses signal curve)	Long curve: # of samples in MA, I = (2 to N) Short curve: (2 to i) for each long curve

		produced Signal curve: # of samples in MA (2 to N)
CCI	Commodity Channel Index (CCI line crosses upper or lower thresholds)	CCI curve: # of samples in MA (2 to N) Upper threshold: 30 to 300 Lower threshold: -30 to -300

These particular indicators were chosen as they span a range of complexity, from one to three dimensions of optimization. We believed this would give a good indication of the benefit of a GPU-based approach for the various indicators available.

Indicator Buy/Sell Signal Descriptions

The EMA indicator is simply an exponentially-weighted moving average of the last N values and produces buy/sell signals as the market data crosses the moving average (MA) curve. The signal may be produced when only the close value crosses the MA (as in EMA-S) or when both the open and close prices cross (as in EMA-OC). The EMA-D indicator uses both a 'fast' and 'slow' moving average curve ('fast' means N is relatively small, 'slow' means N is relatively large), producing buy/sell signals as these curves cross each other.

The MACD indicator subtracts a pair of fast and slow moving average curves to produce the MACD curve that is then smoothed using another moving average to produce the signal line. Buy/sell signals are produced when the MACD and signal lines cross.

The CCI indicator compares the typical price of the currency to its moving average and divides this value by its mean deviation to produce values that oscillate about zero. Buy/sell signals are produced when the CCI crosses a positive or negative threshold above or below zero.

Methodology

In order to assess the accuracy of the GPU algorithms as well as the performance gain, identical indicator optimization routines were written for the CPU. The output of the CPU & GPU runs compared both the optimal parameter(s) and the optimal profit/loss found to ensure that they matched (accuracy check), as well as comparing the total time taken to perform the calculations (to assess performance improvement).

Deviations from Initial Project Plan

The original project plan included the implementation of the arg max routine (to search the array of profit values to find the index of the maximum profit) on the GPU, but it was found that the arg max computation time was only a very small fraction of the overall algorithm time (<2%). We therefore felt that there was not enough benefit to implementing this on the GPU and left it as a simple CPU-based algorithm instead.

The original plan also called for implementation of Simple Moving Averages (SMA) and Linear Weighted Moving Averages (LWMA). However, we decided to implement the MACD and CCI indicators instead, to assess performance on more lengthy computations.

GPU Implementation

Overview

The GPU optimizations were performed using several different sets of open-high-low-close (OHLC) data for a specific currency pair over a specific time period. Each indicator was optimized using a separate kernel for that indicator. These kernels output an overall profit/loss value by obeying each breakout point generated by the indicator into an array with the index of the highest value indicating which thread (and ultimately which set of input parameters) achieved the best result – this constituted the optimization output for each indicator.

Shared memory loading

For every indicator, the first step is loading the relevant OHLC data from Gmem into Smem. We only load the part we need: usually Close and Spread. Loading is done cooperatively. Because all of this data is going to be used by all threads in a block, arithmetic intensity is high, so loading cost is negligible.

Parameter distribution

For the EMA indicators (EMA-S, EMA-OC, and EMA-D), every thread evaluates a different parameter setting (i.e. an N parameter for EMA-S and EMA-OC, and an N_{long} & N_{short} parameter pair for EMA-D). We ensure that the parameters (or parameter pairs) of every warp are close to each other, so that branching is unlikely to diverge. For EMA-S and EMA-OC, we had blocks of just one warp, so that there was at least one block for each multiprocessor. For EMA-D, we had blocks of 512, to save on the cost of Smem loading.

For MACD, every block evaluates an N_{long} & N_{short} parameter pair, optimizing over N_{signal}. That allows more optimization through pre-computing.

MACD

For MACD, our original approach of having each thread evaluate a different parameter triple (N_{long}, N_{short}, and N_{signal}) ran into memory problems for the bigger runs, but wasn't efficient anyway. MACD computation can be optimized by pre-computing the MACD values for a given N_{long} & N_{short} pair, before running the simulation with various N_{signal} values. To allow this, we decided that every block takes an N_{long} & N_{short} pair, and internally optimizes over N_{signal}.

$MACD(N_{long}, N_{short}, i) = EMA(N_{short}, i) - EMA(N_{long}, i)$. EMA has to be computed sequentially, so the naive implementation of MACD pre-computing has to be done by a single thread. That naive implementation took 10% of the total compute time. It is more efficient to pre-compute EMA for all N values, using a separate kernel, and to store those EMA sequences in device memory. Then, $MACD(N_{long},$

N_{short}, *) can be pre-computed by all threads together, which takes only 1% of the total compute time (probably spent mostly waiting for Gmem).

After the threads of a block finish the simulations for all possible N_{signal} values, an argmax is needed. Having a single thread perform this argmax took 10% of the total time. More efficient is to have all threads of the first warp calculate argmax over a subset of the values. Afterwards, the first thread does the maximization over the 32 partial results. Both of those steps take .6% of the total compute time, for a total of 1.2%, i.e. much better than 10%.

Another issue with MACD is the amount of shared memory needed, for each block. We need four arrays, each of 1000 or 1043 values for the biggest data sets: macd[], close[], spread[], and profit[]. Profit[] is used for the result of the trading simulations using the various N_{signal} values. Four float arrays of that size, plus a few smaller pieces of data in Smem, is too much (Smem can hold only 4k float values). To remedy this problem, we collapsed close[] and spread[] into one array, called price[]. When the MACD value is increasing, the only transaction that could possibly result is a BUY. In those cases, price=close+spread/2. When MACD decreases, only a SELL is possible, so price=close-spread/2. Thus, only three arrays are needed: macd[], profit[], and price[]. Loading only a few values at a time, as it is done by the matrix multiplication algorithm, is more complicated, because some threads have to run more than one simulation.

CCI

After we wrote a simple CPU implementation for CCI, we started thinking about how to run it efficiently on a GPU. After discovering that more and more could be pre-computed, we eventually realized that a more drastic optimization is possible. *The optimal upper cutoff value is independent of the lower cutoff value, and vice versa*. In other words, the optimal upper cutoff value will be the same for a lower cutoff of -100, -200, -90, or really anything else. Thus, the optimal upper cutoff can be calculated without considering the lower cutoff value, and thus without a loop over lower cutoff values. This reduces the loop nesting level from three (N_{smatp}, UpperCutoff, and LowerCutoff), to two (N_{smatp}, UpperCutoff). In the same way, the optimal lower cutoff value can be calculated without considering the upper cutoff values. The profit of the entire trading simulation is then the profit resulting from purchases based on the upper cutoff, plus the profit resulting from purchases based on the lower cutoff.

This optimization resulted in a 250x speedup, without using a GPU. After applying the optimization, the resulting run times were all under one second, so it became meaningless to further optimize using a GPU.

Evaluation

The indicator analysis used daily Open-High-Low-Close (OHLC) data from 2005/01/01 to 2008/12/31 (1043 samples) for the EUR/USD currency pair. The first 43 samples were used to prime the

moving averages in indicators that involved them while the final 1000 samples were used to perform the buy/sell analysis. All tests were conducted on the ug64.eecg.utoronto.ca machine.

EMA Indicator Results

For the EMA indicator, the results of the GPU and CPU analysis of the input data are as follows:

Table 2 – EMA-S Results

	Optimal N	**Max Profit**	**Time (ms)**
CPU	104	$46,990	5.251
GPU	104	$46,990	0.643
		Speedup (x):	8.2

Table 3 – EMA-OC Results

	Optimal N	**Max Profit**	**Time (ms)**
CPU	290	$35,650	5.512
GPU	290	$35,650	0.676
		Speedup (x):	8.2

Table 4 – EMA-D Results

	Optimal Long N	**Optimal Short N**	**Max Profit**	**Time (ms)**
CPU	122	49	$47,450	2560.91
GPU	122	49	$47,450	34.363
			Speedup (x):	74.5

We can see that both GPU and CPU produced matching results. Additionally, the GPU achieved a speedup of 8x for EMA-S/OC indicators and an impressive 75x for EMA-D compared to the CPU implementation. The optimal EMA curves plotted against the closing price of the input data are shown in the graphs that follow.

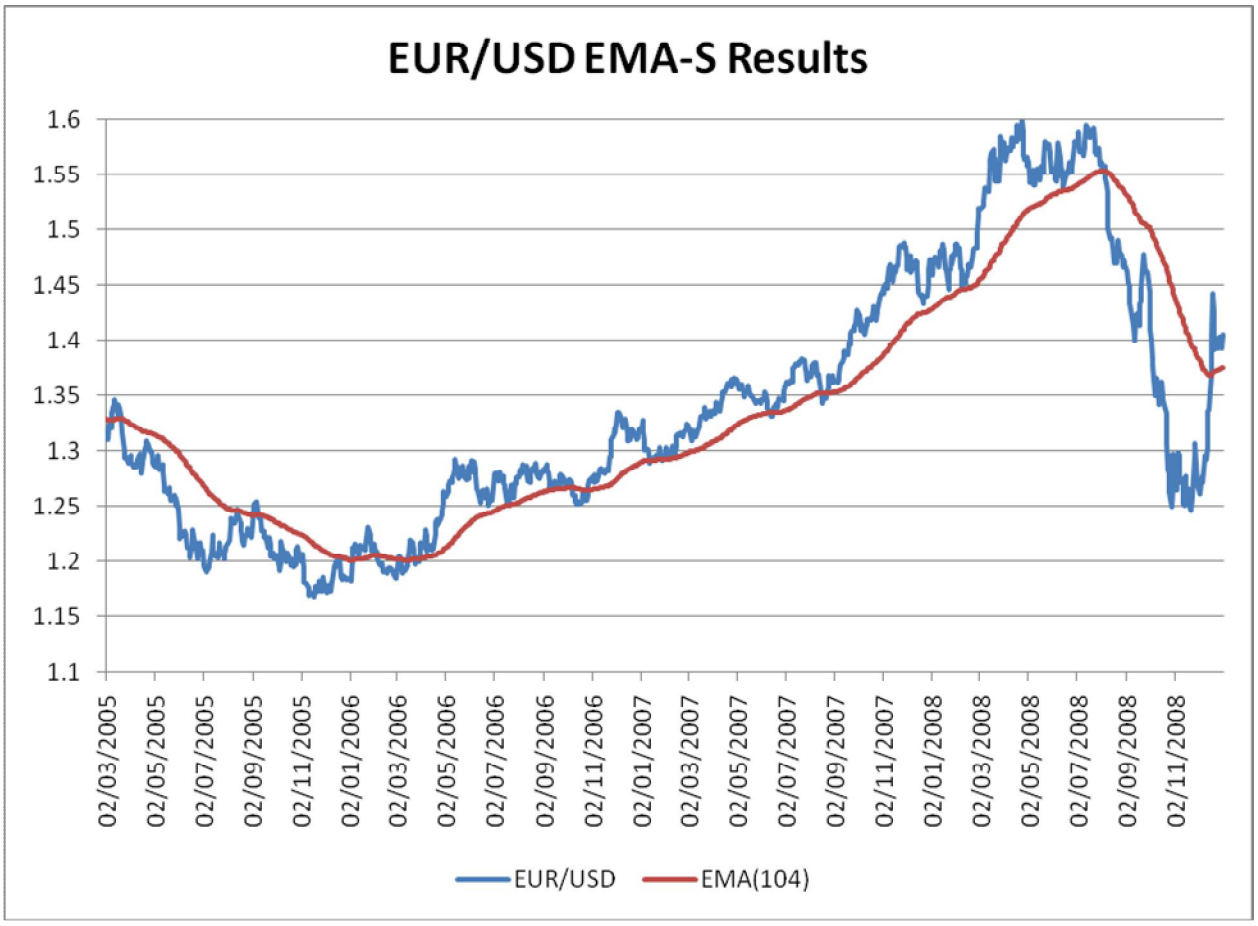

Fig. 2 – EMA-S Result Curve against Close Data, Optimal N = 104 samples

The EMA-S indicator produces a BUY signal when the close price crosses the moving average curve in an upward direction and a SELL signal when the close prices crosses in a downward direction. In the chart above, the optimal 104-sample EMA curve closely tracks the market uptrend and downtrend. Critically, the indicator produces a sell signal on August 6, 2008 when the close price crossed below the EMA curve, predicting the coming downtrend.

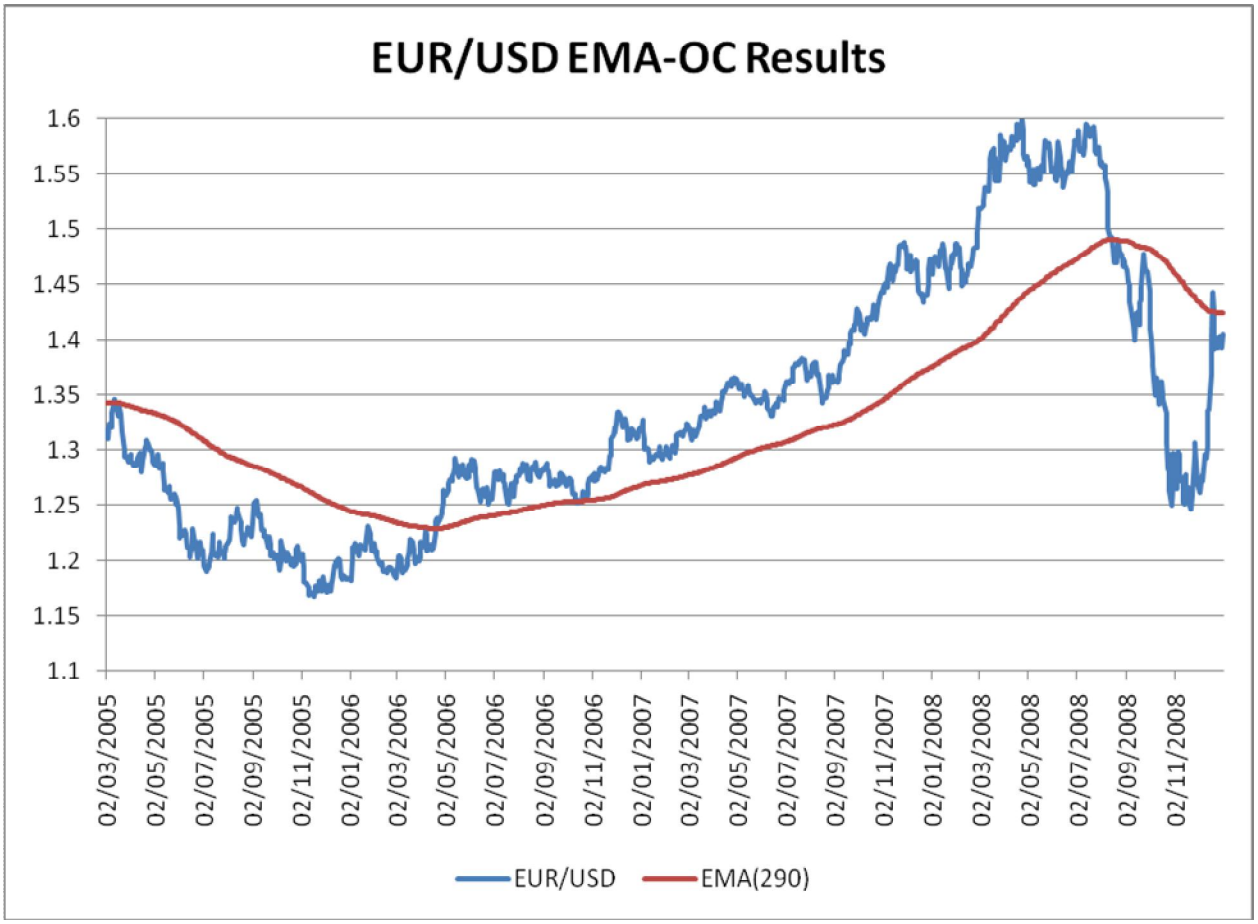

Fig. 3 – EMA-OC Result Curve against Close Data, Optimal N = 290 samples

The EMA-OC indicator is identical to EMA-S except that both the open and close prices must cross the EMA curve in order for a buy or sell signal to be produced. The optimal 290-sample EMA curve initially takes a sell position (tracking the initial downtrend) followed by a buy signal on April 20th, 2006 that predicts the coming uptrend.

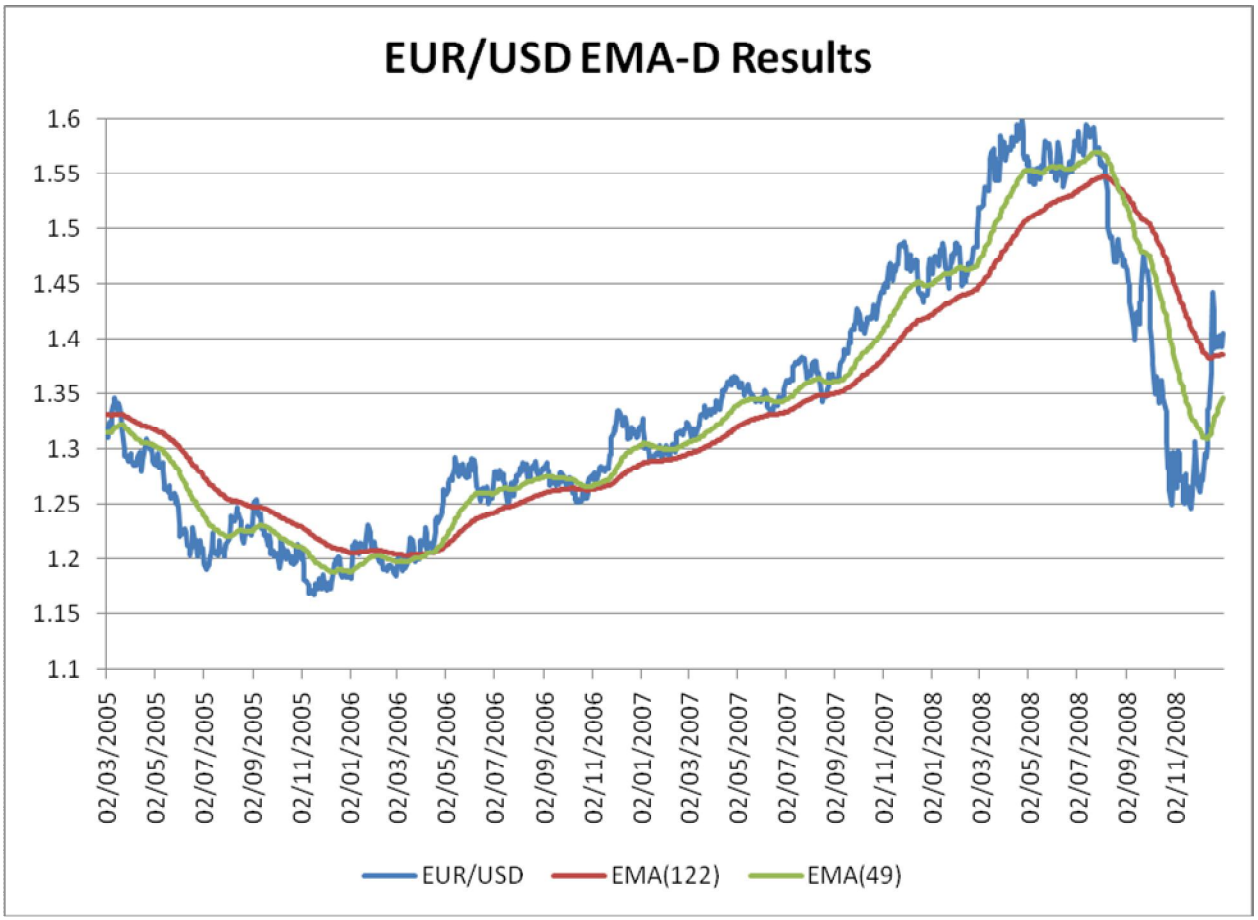

Fig. 4 – EMA-D Result Curve against Close Data, Optimal N_{long} = 122 samples, N_{short} = 49 samples

The EMA-D indicator chart produced only two breakout signals, first a BUY when the 49-sample EMA crossed the 122-sample in an upward direction on April 8, 2006 and a second SELL signal when the cross occurred in the opposite direction on August 22, 2008, also closely tracking the market trend.

MACD Indicator Results

The results for the MACD indicator over the same analysis period are as follows:

Table 5 – MACD Results

	Optimal Long N	Optimal Short N	Optimal Signal N	Max Profit	Time (ms)
CPU	121	119	171	$66,550	2395032.75
GPU	121	119	171	$66,550	18041.26
				Speedup (x):	132.8

Again, the CPU and GPU results match, and the GPU shows a large speed gain of 133x. The benefit of the GPU is much more apparent when optimizing the MACD indicator because of the large number of optimizations that were made in the GPU implementation of the indicator algorithm. The optimal MACD and signal curves plotted against the closing price of the input data are shown below:

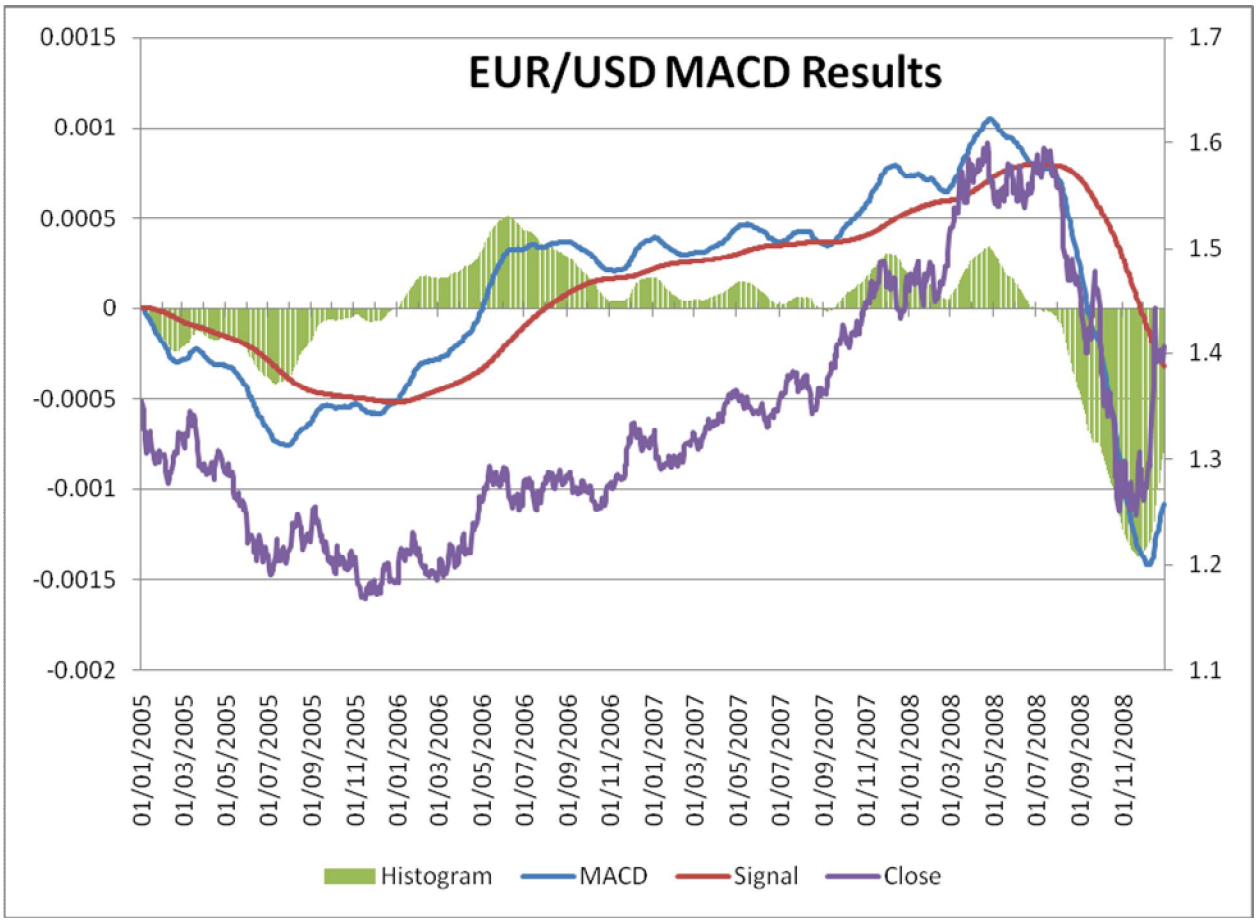

Fig. 5 – MACD Result Curves against Close Data, Optimal N_{long} = 121 samples, N_{short} = 119 samples, N_{signal} = 171 samples

The graph above shows the MACD curve (in blue) plotted against the signal curve (in red), with the subtraction of these two plotted as a histogram (in green) on the primary axis (the display of this histogram is standard when viewing MACD in technical analysis software but was not used in our analysis). The closing prices (in purple) are plotted on the secondary axis. A buy signal is produced when the MACD curve crosses above the signal curve and a sell signal when the curves cross in the other direction. Only a few such signals were produced, most critically a buy signal on December 30, 2005 and a sell signal on June 28, 2008 which were both well timed with the coming uptrend and subsequent downtrend of the market close data.

CCI Indicator Results

The results for the CCI indicator over the same analysis period are as follows:

Table 6 – CCI Results

	Optimal N	Upper Threshold	Lower Threshold	Max Profit	Time (ms)
CPU	43	38	-127	$46,010	385914.81
CPU2 (optimized)	43	38	-128	$46,010	2285.39
				Speedup (x):	168.9

From the table above, we can see that the initial CPU algorithm and the highly optimized CPU algorithms produced nearly identical optimal values, except that the lower threshold value differed by one. However, since the optimal profit values produced by each were the same, the difference in this threshold value has no impact on the outcome and can be ignored. The speedup of the optimized CPU implementation versus the original implementation was 169x. The optimal CCI curve and upper/lower thresholds plotted against the closing price of the input data are shown below:

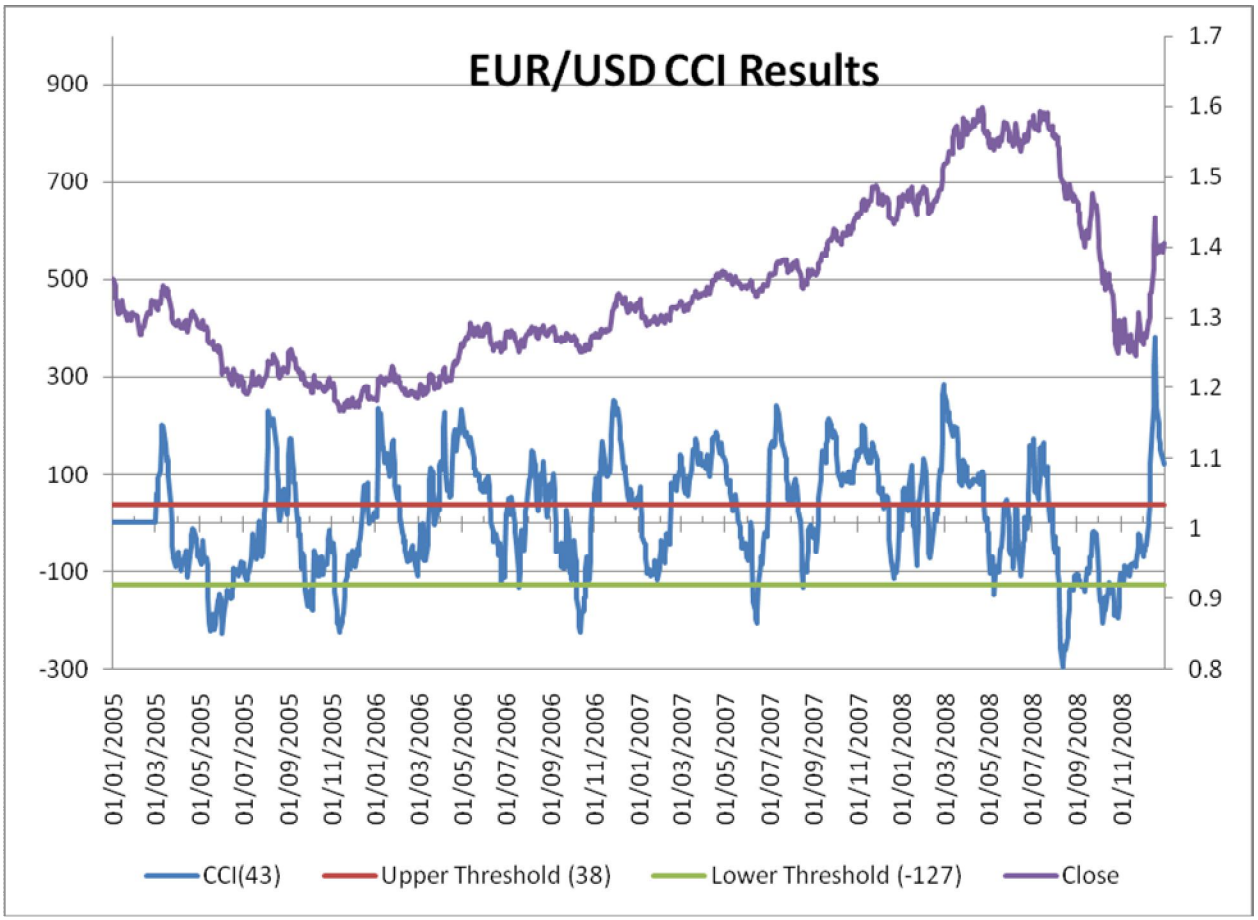

Fig. 6 – CCI Result Curve and Upper/Lower Thresholds against Close Data, Optimal N = 43 samples, Upper Threshold = 38, Lower Threshold = -127

The graph above shows the CCI curve calculated using a 43-sample moving average (in blue) along with the optimal upper threshold value (CCI = 38, shown in red) and lower threshold value (CCI = -127, shown in green). The closing prices (in purple) are plotted on the secondary axis. A buy signal is produced when the CCI curve crosses above the upper threshold, and this position is closed when it crosses back below this threshold value. Similarly, a sell signal is produced when the CCI crosses below the lower threshold, and this position is closed when it crosses back above this threshold value. There were a large number of buy and sell signals produced using these threshold values with most of the profit being produced in the final year of the analysis interval. Although these values produced the optimal profit over the interval, this optimal profit ($46,010) was lower than most other indicators used in this project as some of the intermediate signals produced were false positives and would have lead to a loss of invesment capital.

Conclusions

Use of the GPU was found to have a significant benefit in the optimization of Forex technical analysis indicators. For simple indicators (EMA-S/EMA-OC) a performance gain of 8x was obtained, but for more complex indicators, the benefit was significantly greater – 75x for EMA-D and 133x for MACD. If incorporating the optimization of charting indicators as part of a real-time trading platform, use of the GPU would be highly recommended at least for more complex indicators. In the case of MACD for example, the time required to perform a full optimization of the input parameters was reduced from 40 minutes on a CPU to a mere 18 seconds on the GPU.

A secondary result is that it is important to fully understand the optimization parameters of each indicator as even seemingly complex indicators may have efficient CPU implementations. In the case of CCI, for example, we were able to speed up the CPU implementation by 169x after realizing that the upper and lower threshold values could be optimized independently.

This project also found that the GPU is well suited for performing brute-force optimization algorithms as the input to each run of the optimization routine in a brute force approach is independent of the results of other iterations. For such applications (technical indicator analysis being one of them), it is often realistic to expect a high performance gain using a GPU-based algorithm implementation.

A logical extension of this work would be to implement more technical analysis indicators and combine the results into a real-time trading platform, or to apply a similar brute-force optimization approach to problems where trying every combination on a CPU may take many hours or days.

Works Cited

Janssen, C., Langager, C., & Murphy, C. (2009). *Technical Analysis: Moving Averages*. Retrieved March 7, 2009, from Investopedia: A Forbes Digital Company: http://www.investopedia.com/university/technical/techanalysis9.asp