Fast Business Decisions with Parallel Computation

Dr. John Jenq

Dr. Hubert Johnson

Department of Computer Science
Montclair State University
Montclair, New Jersey, 07043
USA

Abstract

In recent years, make quick decision in business become more and more important. Companies adopt fast parallel machines to achieve the goal of making quick decisions to compete with their competitors. Devices such as field programmable gate arrays (FPGA), graphics processing unit (GPU) were used in the area such as high frequency trading. The embedded GPU on computers and the decline of personal computer prices make parallel processing affordable with the price of even personal computers. In this paper, we will investigate the computational aspects of currency swapping and moving averages. For currency swapping, there are arbitrage opportunities when there exist price difference among different currencies in the international exchange market. The major process of the arbitrage strategy is to convert one currency to another, then convert it again to a third currency, etc. and, then eventually convert it back to the original currency within a short time span. The simplest form of this process is the triangular arbitrage which involves only three currencies. It is possible to process more than three currencies in order to apply the arbitrage process. Although it may seem as if arbitrage is a risk-free process, in reality there are risks involved. One of the risk factor is time. I.e., how fast one can process and how much capital one has in order to take advantage of these price discrepancies. It is worth noting that the arbitrage opportunities only exist when a bank's quoted exchange rate is not equal to the markets implicit cross exchange rate; all these happen in the range of seconds. The high frequency market fluctuation makes this process a challenge and call for the utilization of high-performance computation. In this paper, we investigate the arbitrage problem utilizing GPU to identify the opportunities and speed up the process. For Moving average, we will focus on the parallel computation of simple moving average and exponential moving average operations, two of the most popular financial indicators. For currency swapping we will discuss the factors which involve in the arbitrage process. In this paper, we also investigate the usage of GPU to run artificial neural network as a mean to predict stock market pricing. Feed forward and back propagation artificial neural network was used for this study. Financial data including major stock indices, volumes, pricing, and moving average of stocks were used as input. The future stock prices can be predicated as the output. The speedup factor by adopting GPU and CPU together over traditional CPU alone implementation was not significant. The computation of compute moving averages and currency swapping on GPU also discussed.

Keywords: Artificial neural network, stock prediction, GPU computing, parallel processing, high performance computing. Moving averages, currency swapping.

Introduction

Graphic processing unit (GPU) has transformed a regular PC to a personal supercomputer. For example, the GeForce GTX 580 can perform single precision operation that reaches more than 1500 GFlops. These computing powers significantly speed up the computational intensive applications with a price of a PC. Many tools have been developed to make the GPU computing much easier than ever before. Personal supercomputing is now a reality to us. Artificial neural networks are used for pattern recognition, clustering, and optimization. Neural networks can also be used to solve problems which are not easily solved by tradition calculation methods, particularly, if there is no strong underlying theory to explain the data.

Neural networks have been developed as generalizations of mathematical methods of neural biology, based on the assumption that the information processing occurs at many simple elements called neurons. Signals are passed between neurons over connection links. Each connection link has an associated weight which multiples the signal transmitted. Each neuron applies an activation function to its net input to determine its output signal. In the world financial market today, there are many exchange rates. Banks have their own exchange rate posted and these rates are dynamic in nature. It is a possible that these exchange rates is not consistent during a short period of time but they will go back to so call equilibrium state. It is therefore raised a question of currency arbitrage. Traditional, arbitrage of currency swap occurs when the interest rates of two parties in two countries are different. By swap the currency from one to the other and the save the money to other country may give opportunity to profit more. This practice usually comes with a forward contract to safe guard the outcome. It is sometimes used to hedge assets as well. Currency swaps are over-the-counter derivatives that serve two main purposes. First, they can be used to minimize foreign borrowing costs. Second, they could be used as tools to hedge exposure to exchange rate risk.

There are arbitrage opportunities when there exist price difference among different currencies in the international exchange market in a very short period of time such as seconds. The major process of this kind of arbitrage strategy is to convert one currency to another, then convert it again to a third currency, etc. and, then eventually convert it back to the original currency within a short time span. The simplest form of this process is the triangular arbitrage which involves only three currencies. It is possible to process more than three currencies in order to apply the arbitrage process. Although it may seem as if arbitrage is a risk-free process, in reality there are risks involved. One of the risk factor is time. I.e., how fast one can process compare with your competitors and how much capital one has in order to take advantage of these price discrepancies. It is worth noting that the arbitrage opportunities only exist when a bank's quoted exchange rate is not equal to the markets implicit cross exchange rate; all these happen in the range of seconds. The high frequency market fluctuation makes this process a challenge and call for the utilization of high-performance computation. In this paper, we investigate the arbitrage problem utilizing GPU to identify the opportunities and speed up the process. Park investigate features of the currency swap pioneered by the World Bank in 1981 (Park 1984).Goswami et.al. Investigate the reasons why multinational firms continue to use currency swapping (Goswami 2004, 2007).

Usmen create a model to measure the gains of currency swapping on international market (Usmen, 1994). Kozhan and Tham investigate the risk of high frequency arbitrage (2012).They demonstrated the risk come from crowding effect of competing arbitrageurs entering the same trade and inflicting negative externalities on each other. Doukas et.al. Examine the relation between equity mispricing and arbitrage risk. They found stocks with high arbitrage risk have higher estimated mispricing than stocks with low arbitrage risk. They concluded that mispricing is a manifestation of the inability of arbitrageurs to hedge idiosyncratic risk, a major deterrent to arbitrage activity. Researchers have adopted neural network in financial and economic computations. For example, Li and Liu used LM BP algorithm to predict Shanghai stock market (Li 2009). Wang developed an HLP method that gets stock high low point with different frequency and amplitude. Those extracted data are then fed into a neural network to forecast the stock direction and price (Wang 2011). Tirados and Jenq used neural network to predict GDP with ten leading economic indicators as input (Tirados 2009). Lin and Feng combined neural network and pattern matching techniques to analyze and to forecast oil stock price (Lin 2010). Zhou and Zhang used financial indicators such as moving averages, volumes, Relative strength index, etc. on neural network to predict future stock price (Zhou 2010). In the past, CPU clusters have been used to achieve high performance computation. GPU computing uses GPU as a co-processor to accelerate CPUs for general purpose scientific and engineering computing.

It shifts computation intensive program segments into GPU while keeping the rest of the program segments which are serial in nature on the CPU. This kind of hybrid computing improves the performance of many computer applications. The GPU computation can be used on financial computation as well. Researchers in the financial world find the benefits of using GPU in financial computation. Peng, et. al., compute option pricing on GPU with backward stochastic differential equation (Peng 2011). Lee, et. al., did financial derivative modeling using GPUs (Lee 2009). Solomon et. al., used trinomial lattice strategy to implemented the pricing of European option and American look back option pricing using GPU (Solomon 2010). Lee, et. al., investigated random number generation and Monte-Carlo simulation to predict future stock prices. They also discussed the out of core case when graphics DRAM is not big enough to hold all the application data (Lee 2010).The rest of the paper was organized as the following. Section 2 discusses methodologies and implementation of an artificial neural network.

Parallel implementation of currency swapping, simple moving average and exponential moving average will be discussed. Section 3 discusses and analyses the experimental results. Section 4 gives conclusion remarks.

Methodologies and implementations

Three layer neural networks were chosen to implement in our prediction system. The input are major industrial stock indices and stock indicators. The goal is to forecast future stock price. Feed forward and back propagation neural network was used. Back propagation is a gradient descent neural network method used to minimize the total squared error of the output calculated by the network. The network is developed to achieve a balance between the ability to respond to the input patterns that are used for training and the ability to give good responses to input that is similar, but not identical, to that used in training. Like with multiple regressions, back propagation was used to develop a correlation between the inputs of stock market data in order to determine the future stock price. The training of a network by back propagation involves three stages: the feed forward of the input training pattern, the calculation and back propagation of the associated error, and the adjustment of the weights. After training, application of the network involves the computations of the feed forward phase only. To prevent the larger data to dominate the outcome, the raw data will be processed before being fed into the neural network. The raw data has been modified. In the following discussion, B represents the raw form of the original data, and C is the normalized version of B.

$$C_t = 2\left(\frac{B_t - B_{\min}}{B_{\max} - B_{\min}}\right) - 1 \qquad (1)$$

This transformation map our data to between -1 and 1. The activation function selected for this project is the bipolar sigmoid function, which has a range of (-1, 1), and is defined as

$$f(a) = \frac{2}{1 + exp(-a)} - 1 \qquad (2)$$

$$\frac{df(a)}{da} = \frac{1}{2}[1 + f(a)][1 - f(a)] \qquad (3)$$

Feed forward back propagate Neural Network

The traditional Neural network algorithm can be simplified as below while (cycle < max Cycle & average Error > tolerance Eerr)

```
    For (int i= 0; i < total Records;  i++)
    {
        Forward Propagation ();
        Backward Propagation ();
        Accumulate Error;
        Update Weights (learning Rate);
    }
    compute average Err;
}
```

To parallelize the code in order to fit into GPU computing, some modifications will be done. Instead of performing back propagate operation for each pattern to update weights, we will compute the dw (the update of weight for w) and then do the update of the weights at the end of each cycle. It means that we move the update Weights(learning Rate) out of the traditional algorithm above. Instead of performing this operations for each record per cycle, we reduce them to once per cycle. Therefore the total weights to be updated would be the summation of dw and dv which are calculated by individual pattern during the process. Note the dws are the weights to be added to the w, the weights between input layer and hidden layer. And dvs are the weights to be added to V, the weights between hidden layer and output layer. Here we assuming three layers neural network is used, and there is one output neuron. The update Weigths can be done by binary reduction which can be performed in logN steps using N threads. Although the backward Propagation and update Weights may be parallelized but the gain of speed up may be limited. This setup allows us to assign one pattern to each thread and therefore speed up the process. Because the reduction operation may slow down the whole system performance, it is interesting to find out if assign more than one pattern to a thread to process will improve the performance. Once again, it depends on how much clock cycles will be used to synchronize the threads.

The more threads to be synchronized, we expect more time to do the reduction. Interesting thing is to find out how to organize so that we can get the best possible performance.

Moving Average

Moving averages can be used as financial indicators. There are various type of moving averages. The two most popular moving averages are simple moving average and exponential moving average.

Simple Moving Average

Assuming the daily closing price for day *t* is C_t. The *n*-day simple moving average can be defined as $SMA(t) = \frac{\sum_{i=t-n}^{i=t-1} C_i}{n}$, where C_i is the closing price at day *i*. So simple moving average can be computed by taking the average closing price of a stock, over the last *N* periods. Popular simple moving averages are 5, 10, 20, 40, and 200. Let's assume the last five periods for a stock are 1,3,5,7, and 9, then the 5 day simple moving average can be computed as (1+3+5+7+9)/5 = 5. While simple moving average giving all past *n*-day closing price equally weight, the *n*-day exponential moving average assign more weight to recent price as will be discussed in the next subsection. Simple moving average on a parallel computer can be done by using Prefix Sum operation. It also known as Scan operation. A Prefix Sum can be defined as the following. Given a set of *N* values $a_1, a_2, a_3, \ldots, a_n$, and an associative operation @, the Prefix Sums operation will compute the *N* quantities (a_1 , $a_1@a_2$, $a_1@a_2@a_3$,..., $a_1@a_2@a_3@ \ldots @a_n$). By using the prefix sum operation, the *N*-day Simple moving average of day *"i"* can be calculate as

$$SMA[i] = (prefixSum[i] - prefixSum[i-N] \quad (4)$$

In Eq. (4) *SMA[i]* stands for the *N* day moving average at day *i*. The Table 1 gives an example that using prefix sum to compute 3-day moving average. Assume the missing period (period -1, and -2 in our example) values are 0.

Period	1	2	3	4	5	6	7
Value	1	3	5	7	9	10	12
prefixSum	1	4	9	16	25	35	47
Total of subsequence	1	4	9	15(16-1)	21(25-4)	26(35-9)	31(47-16)
Simple Moving Average	1	2.5	3	5	7	8.67	10.33

Table 1: Example of calculation of SMA using prefixsum

Exponential Moving Average

Exponential moving average can be defined as $EMA(t) = (P_t - EMA(t-1)) * \alpha + EMA(t-1)$, where P_t is closing price at day *t*. The α is weighting factor and can be defined as $\alpha = 2/(n+1)$, where *n* is the number of time periods involving in the computation. For example, for 10-day *EMA*, $\alpha = 2/(10+1) = 0.181818$. While 20-day *EMA*, $\alpha = 2/(20+1) = 0.095238$. One can rewrite the above *EMA* definition using past *n*-day closing prices and α as follows. Note weights are decayed exponentially and the most recent prices carry more weights in the computation.

$$EMA(t) = \alpha P_t + \alpha(1-\alpha)P_{t-1} + \alpha(1-\alpha)^2 P_{t-2} \ldots + \alpha(1-\alpha)^{n-1} P_{t-n+1}$$

Although *EMA* can be easily computed by using previous *EMA* as mentioned in the previous paragraph, but this has serial nature in the computation. In order to deliver a parallel algorithm, let's define $\beta = (1-\alpha)$, where α as defined from the previous discussion.We will partition the whole data array into segments with length of power of 2. Assume the leftmost data is the most recent data (the lowest index) and rightmost one (the highest index) is the oldest timing data. By using $O(N)$ memory to store the temporarily information, we can compute exponential moving average of P period in $O(log_2 P)$ time. Here N is the number of records. Actually one can reduce the total memory to $2N$ as we will discuss shortly. The computation of *EMA* involves two phases. In the first phase, we will generate the temporarily data through iterations. In the first iteration, two pieces of data which are adjacent to each other will be processed to create combined length-2 information. In the second iteration, combined length-4 segment information can be generated. To simplify, let's assume p is power of two. In $log_2 P$ iterations, one can create $\frac{P}{2} + \frac{P}{4} + .. + 1$ with total of $P-1$ pieces of data information for each segment of length *P*. These information will be used in the second phase of our *EMA* computation. The Table 2 gives an illustration of the first phase *EMA* computation. Note $\beta = (1-\alpha)$.

0	1	2	3	4	5	6	7
	(01)		(23)		(45)		(67)
		(0123)				(4567)	
				(1->7)			
	$C_0 + \beta C_1$	$C_0 + \beta C_1 + \beta^2(C_2 + \beta C_3)$	$C_2 + \beta C_3$	$C_0 + \beta C_1 + \beta^2(C_2 + \beta C_3) + \beta^4(C_4 + \beta C_5 + \beta^2(C_6 + \beta C_7))$	$C_4 + \beta C_5$	$C_4 + \beta C_5 + \beta^2(C_6 + \beta C_7)$	$C_6 + \beta C_7$

Table 2: Phase one of EMA Computation: data store scheme

For a segment of *P* periods of data on a *N* data array, where $P < N$, our job is to find the *EMA* for all the data on the data array with *N* data. Our approach is to partition any segment from index *a* to index *b*, where (*b-a+1*) is the period *P*, into at most two segments. Let's assume there are two segments, P_1 and P_2. The first segment P_1 starts with index *a* and ends with an index which is power of 2. The second segment P_2 starts with an index which is power of 2 and ends with *b*. Note segment P_1can be formed by elements with length of power of two. For example, if segment $P_1$is of length 11, then it can be formed as sum of segments of lengths 1, 2, and 8. Note here the lengths of the segments from left to right are in increasing order. P_2 can be processed similarly. But the component segments size are in decreasing order. For example, if P_2 is of length 14, then the segments will have length 8, 4, and 2. Note also these component segments never carry same length as other segments in its combination. We can use a loop to mark the segments if they are not zero and then sum these up to get the resultant EMA. Note during the computation, different power of β need to be used to faithfully reflect the weights assigned to these segments. Note the index scheme which we generated previously in phase one shall be used to retrieve information. For example, for a segment index 54 to 63. It can be partitioned into two sub-segments one from index 54 to 55(length 2) and the other from 56 to 63 (length 8). Consider the segment of length 8, we can retrieve information at index $\lceil (56 + 63)/2 \rceil = 60$ as described in phase one from previous paragraph. The content with index 60 contains information of indices from 56 to 63. Note when we do addition, this value need to be multiplied by β^2. The square is due to the fact that the segment 54 to 55 has length of 2. We omit the details of the parallel algorithm here.

Currency Swapping

We collect For Ex tick data from Internet. There are total of ten currencies that are under consideration: AUD, CAD, CHF, EUR, GBP, JPY, NZD, TRY, GSD, and USD. These data were preprocessed and extracted to form a single data file, which contains date, and time, bid and ask prices, and base and quote currency IDs. We assign each currency an integer ID to facilitate the program processing. There are two programs involved. One is serial and the other parallel counterpart.

Serial implémentation

The serial program takes each individual tick data and updates the data structure. Each tick only involves two pieces of data, one for base currency and the other for quote currency. As soon as the data was processed, a signal will be raised if there is an arbitrage opportunity that occurred when the data structure was updated. The information will then be output onto a data structure that can be output to a file at the end of this simulation.

Note that for each tick data process, the pair of data only affects a portion of the data structure. We represent the exchange rates as a multiplier, as shown in Figure 1. For example, to compute the result of converting currency A to B, B to C, then from C back to A, we will compute $\boldsymbol{M_{AB} * M_{BC} * M_{CA}}$, as shown in Figure 1. Note that we ignore all the transaction costs which were involved in the currency exchange processes

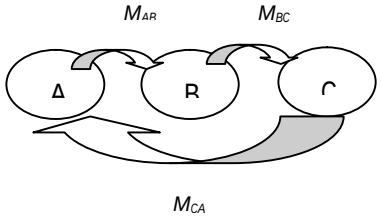

Figure 1: Compute the possibility of currency swapping arbitrage

2.3.2 Parallel arbitrage Algorithm

Because currency swapping has data dependency due to the fact that tick data are in time series, both the time sequence nature and the current tick data will affect and change the data structure. Therefore, in order to speed up the process, we spread the work which is supposed to be performed in each tick to as many threads as possible. Based on this observation, we assign each thread to process one operation which will check possibility of arbitrage. The next tick data can be accessed by all devices and will be put into shared memory by a particular device with a mapped device ID. Because only ten currencies were considered and since we only concerned about triangular currency swapping, it can be proved that there are at most twenty operations needed to be considered each time there is a tick. We don't have to consider all 1000 operations because many of them remain the same.

Assuming a tick data contains a base currency *b* and quote currency q, with bid and ask prices given, we only need to consider the operation of$M_{bi} * M_{iq} * M_{qb}$, for $0 < i < 10$. Similarly, we have to perform $M_{qi} * M_{ib} * M_{bq}$., for $0 < i < 10$. Since there are 1000 possible permutations, we will allocate 1000 threads for our application. A thread with *tid* of b_{tq} will process $M_{bt} * M_{tq} * M_{qb}$ each time base currency *b*, quote currency *q* (or reversed when base currency *q* and quote currency *b* tick data were received). For example, if the *tid* of a thread is 983, then when base currency $b = 9$, quote currency $q = 3$ were received, it will compute the value of $M_{98} * M_{83} * M_{39}$. Because this assignment is unique, there is no conflict among threads. This assignment will reduce the 20 fold host operations to one operation in each thread.

Experimental Results

The experiments were conducted on an Intel i7 with Nvidia GeForce 550M, a low end GPU machine. The parallelized moving average and neural network version was constructed using CUDA C. Due to the small size of records and the nature of the neural network, the speed up wasn't observed. The main reason is due to the requirement of synchronization of these threads. It is apparent that the expensive cost of synchronization makes the CPU implements more appealing. We expect when the number of neurons and number of data increase, we shall get better results. For currency swapping when no transaction overhead of currency swapping were involved, we might get 2% incidents of possible arbitrage. But these transactions must be carried out in seconds. When the overhead fee of transaction rose to say 1% then the percentage of arbitrage opportunity by swapping currencies were reduced to 0. I.e., it is impossible to have financial gain by swap currency in very short time. Consider 3091 ticks; there are 89 possible arbitrages when zero overhead was considered. When there is 1% overhead, the opportunity reduces to 0. The running time to process the tick data in host computer took average of 22 ms, while the parallel counterpart only achieves 17 ms with speed up less than 2. The speedup factor of using GPU in currency swapping is therefore minimal. There is no significant gain of using parallel implementation to compute currency swapping arbitrage. One of the reason is the syncthreads() operation which is used to avoid the so call read-modify-write race condition constraint. The other reason is due to the processing speed factor of CPU and GPU.

Conclusion Remarks

The parallel versions of currency swapping and moving average computation used in financial industry and back propagation neural network computation were developed to run on Nvidia GPU using CUDA C. The GPU version of moving average does not give significant speedup over traditional CPU version using prefix sum operation. For neural network training process, GPU computation does not provide significantly performance over traditional CPU implementation due to the requirement of thread synchronization.

Even if we tried to minimize the number of synchronization by using device kernel calls but speed up over the traditional CPU approach wasn't significant. Actually in some situations when the number of records is small, the CPU implementation is superior. The implementation of real time predicting system over huge data set in financial industry is an interesting and challenging problem for future investigation. For currency swapping, we expect that if more currencies involved in the computation instead of Just trianguler, the parallèle implémentation will give us better speed up results. In this report, we only used ten currencies, if there are more currencies involve in the exchange, our implementation shall receive better performance. In summary, for currency swapping and moving average computation, GPU computing provide limited advantages over CPU computing when the number of data is small. Increase the number of data by simulation, some improvement can be achieved. We know different machine with different model of CPU and GPU can make the results different. We expect with faster GPU the results will be improved.

References

John A. Doukas, Chansog (Francis) Kim, Christos Pantzalis, (2010), "Arbitrage Risk and Stock Mispricing", The Journal of Financial and Quantitative Analysis, Vol. 45, No. 4 (AUGUST 2010), pp. 907-934

Roman Kozhan, Wing Wah Tham, (2012), "Execution Risk in High-Frequency Arbitrage", Management Science, Vol. 58, No. 11 (November 2012), pp. 2131-2149

Myungho Lee, Chin Hong Chun, and Sugwon Hong, (2009) " Financial Derivatives Modeling Using GPU's", International Conference on Scalable Computing and Communications; The Eighth International Conference on Embedded Computing, pp 440 – 445

Gautam Goswami, , Jouahn Nam, Milind M Shrikhande, (2004) "Why do global firms use currency swaps?: Theory and evidence", Journal of Multinational Financial Management, Vol 14, Issues 4–5, October–December 2004, pp 315–334

Gautam Goswami and Milind M. Shrikhande, (2007) "Economic exposure and currency swaps", Journal of Applied Finance 2007, Vol. 17, no. 2, pp 1-9

Myungho Lee, Jin-hong Jeon*, Joonsuk Kim, and Joonhyun Song, (2010), " Scalable and Parallel Implementation of a Financial Application on a GPU: with focus on out-of-core case", 2010 10th IEEE International Conference on Computer and Information Technology, pp 1323 - 1327

Feng Li, and Cheng Liu, (2009), "Application Study of BP Neural Network on Stock Market Prediction", 2009 Ninth International Conference on Hybrid Intelligent Systems, pp 174 - 178

QianYu Lin, and ShaoRong Feng, (2010), " Stock market forecasting research based on Neural Network and Pattern Matching", 2010 International Conference on E-Business and E-Government, pp 1940 - 1943

Y. S. Park, (1984) "Currency Swaps as a Long-Term International Financing Technique", Journal of International Business Studies, Vol. 15, No. 3, winter, 1984, pp. 47-54

Ying Peng, Bin Gong, Hui Liu, and Bin Dai, (2011), "Option Pricing on the GPU with Backward Stochastic Differential Equation", 2011 Fourth International Symposium on Parallel Architectures, Algorithms and Programming, pp 19 - 23

Steven Solomon, Ruppa K. Thulasiram and Parimala Thulasiraman, (2010), " Option Pricing on the GPU", 2010 12th IEEE International Conference on High Performance Computing and Communications, pp 289 - 296

Edward Tirados and John Jenq, (2009), "Analysis of Leading Economic Indicator Data and Gross Domestic Product Data Using Neural Network Methods", Journal of Systemic, Cybernetics and Informatics, vol 7, no 4, 2009, pp 51-56

Nilufer Usmen (1994), "Currency Swaps, Financial Arbitrage, and Default Risk ", Financial Management, Vol. 23, No. 2, summer 1994, pp 43-57

Lei Wang, and Qiang Wang, (2011) "Stock market prediction using artificial neural networks based on HLP", 2011 Third International Conference on Intelligent Human-Machine Systems and Cybernetics, pp 116 - 119

Yixin Zhou, and Jie Zhang, (2010), "Stock data analysis based on BP neural network", 2010 Second International Conference on Communication Software and Networks, pp 396 - 399