Comput Econ
DOI 10.1007/s10614-016-9579-y

Parallel Optimization of Sparse Portfolios with AR-HMMs

I. Róbert Sipos[1] · Attila Ceffer[1] · János Levendovszky[1]

Accepted: 29 March 2016

Abstract In this paper we optimize mean reverting portfolios subject to cardinality constraints. First, the parameters of the corresponding Ornstein–Uhlenbeck (OU) process are estimated by auto-regressive Hidden Markov Models (AR-HMM), in order to capture the underlying characteristics of the financial time series. Portfolio optimization is then performed by maximizing the return achieved with a predefined probability instead of optimizing the predictability parameter, which provides more profitable portfolios. The selection of the optimal portfolio according to the goal function is carried out by stochastic search algorithms. The presented solutions satisfy the cardinality constraint in terms of providing a sparse portfolios which minimize the transaction costs (and, as a result, maximize the interpretability of the results). In order to use the method for high frequency trading (HFT) we utilize a massively parallel GPGPU architecture. Both the portfolio optimization and the model identification algorithms are successfully tailored to be running on GPGPU to meet the challenges of efficient software implementation and fast execution time. The performance of the new method has been extensively tested both on historical daily and intraday FOREX data and on artificially generated data series. The results demonstrate that a good average return can be achieved by the proposed trading algorithm in realistic scenarios. The speed profiling has proven that GPGPU is capable of HFT, achieving high-throughput real-time performance.

✉ I. Róbert Sipos
siposr@hit.bme.hu

Attila Ceffer
ceffer@hit.bme.hu

János Levendovszky
levendov@hit.bme.hu

1 Department of Networked Systems and Services, Budapest University of Technology and Economics, Budapest, Hungary

Published online: 07 April 2016

Keywords Mean reversion · Hidden Markov models · Portfolio optimization · Financial time series · High frequency algorithmic trading · GPGPU

1 Introduction

Portfolio optimization was first investigated by Markowitz (1952) in the context of diversification to minimize the associated risk and maximize predictability. Mean reversion is a good indicator of predictability, as a result, identifying mean reverting portfolios has become a key research area (D'Aspremont 2011; Fogarasi and Levendovszky 2013; Sipos and Levendovszky 2013). However, portfolio optimization becomes very complex when introducing cardinality constraints in order to minimize the transaction costs. In this case, optimizing sparse portfolios is proven to be NP hard (Natarajan 1995).

The stochastic nature of a mean reverting portfolio is described by the so-called Ornstein–Uhlenbeck process (the solution of the OU stochastic differential equation). Since, this model is rather limited, we need an extension of the stochastic modeling of portfolio values by using the Auto-Regressive Hidden Markov Model (AR-HMM). AR-HMMs are widely used for predictions (Durbin 1998; Hassan and Nath 2005; Jurafsky and Martin 2006; Mamon 2007), furthermore, a multi-state AR-HMM can be considered as a generalization of the traditional OU modeling (Sipos and Levendovszky 2015), which enables us to perform prediction based algorithmic trading on general financial time series.

So far, optimization of mean reverting portfolios has been performed subject to maximizing the so-called predictability parameter. In this paper we introduce a new objective function, namely, we maximize the return which can be achieved with a predefined probability. This new objective function gives rise to more profitable portfolios.

One of the major changes in the computer software industry made towards high performance simulations has been the move from serial programming to parallel programming. Driven by the insatiable market demand for real-time, high-definition 3D graphics, the Graphics Processing Unit (GPU) has evolved into a highly parallel, multithreaded computational engine (NVIDIA 2014). The GPU is primarily designed for high-speed graphics, which are inherently parallel. The General Purpose GPU (GPGPU) model let the programmer to use this huge computational power for any kind of problems without need for the graphics primitives (Cook 2013).

Using the GPU shows growing interest in the field of computational finance also (Pagès and Wilbertz 2011). There are several applications that have clearly proven the success of highly parallel programming, such as option pricing, risk calculation and time series prediction. In this paper, we will use this computational power for time series modeling, identification and portfolio optimization.

As a result, the main contribution of the paper are given as follows: (i) we maximize the return achieved with a predefined probability based on the corresponding p.d.f.-s instead of optimizing the predictability parameter (which provides us with a higher profit); (ii) we use the AR-HMM approach to model the underlying time series, which is more general than the OU modeling; (iii) the sparse portfolio optimization is carried out by stochastic search yielding a global and valid optimum; (iv) portfolio optimization and model identification algorithms are tailored to parallel architectures such as GPGPU which paves the way towards a more sophisticated HFT.

Finally, trading is perceived as a "walk" in the "buy/sell" action space, which is then tested numerically on randomly generated time series and FOREX rates. The results exhibit good average returns.

The structure of the paper is as follows:

- In Sect. 2, the model and the notations are introduced and the concept of AR-HMM modeling is briefly summarized together with its connection to the mean reverting processes;
- In Sect. 3, we optimize the portfolio according to the new objective function;
- In Sect. 4, the computational model is mapped out;
- In Sect. 5, the parallel implementation details are described;
- In Sect. 6, a detailed performance analysis is given based on historical data;
- In Sect. 7, some conclusions are drawn.

2 Notations and Stochastic Modeling

In this section we describe the model and the concept of mean reverting portfolios, and its generalization (D'Aspremont 2011; Fogarasi and Levendovszky 2013; Sipos and Levendovszky 2015), the Auto-Regressive Hidden Markov Models (AR-HMM).

The time series describing the prices of assets is denoted by $s_t^T = (s_{1,t}, \dots, s_{n,t})$ where $s_{i,t}$ is the price of asset i at time instant t. The portfolio vector is denoted by $\mathbf{x}^T = (x_1, \dots, x_n)$ where x_i gives the number of possessed quantity from asset i. The value of the portfolio at time t is denoted by $p(t)$ and defined as

$$p(t) = \mathbf{x}^T \mathbf{s}_t = \sum_{i=1}^{n} x_i s_{i,t}. \tag{2.1}$$

Our objective is to find the optimal portfolio $\mathbf{x}_{opt}$ which maximizes a predefined objective function, such as minimum return achieved with a given probability (described in Sect. 3), subject to cardinality constraint which specifies that the number of non-zero components in $\mathbf{x}_{opt}$ by being l. This constraint stems from the need for minimizing the transaction cost. A sparse portfolio also implies higher interpretability (D'Aspremont 2011; Becker et al. 2013).

Imposing a sparsity constraint is a common approach in the literature, for example in D'Aspremont (2011), Chang et al. (2000), Fieldsend et al. (2004) or Shaw et al. (2008). This constraint is further motivated by taking into account realistic market scenarios, e.g. if round lots are present (the smallest amount of assets that can be purchased) it would not be feasible to hold dense portfolios consisting only a small number of assets. Another example is, when trading is done on an order book, some of the assets might not be available to buy at the same time or at required quantity.

In the literature, the optimal portfolio is sought under the assumption that the portfolio value $p(t)$ exhibits mean reverting properties and follows an Ornstein–Uhlenbeck

(OU) process (Ornstein and Uhlenbeck 1930). This is a frequent assumption in trading (Fama and French 1988; Manzan 2007; Ornstein and Uhlenbeck 1930; Poterba and Summers 1988) which follows from the VAR(1) nature of the underlying asset process. The OU process is characterized by the following stochastic differential equation

$$dp(t) = \vartheta\,(\mu - p(t))\,dt + \sigma dW(t), \tag{2.2}$$

where $W(t)$ is a Wiener process and $\vartheta > 0$ (mean reversion coefficient), μ (long-term mean) and $\sigma > 0$ (volatility) are constants. One can obtain its solution by using the Itō-Doeblin formula (Ito 1944).

AR-HMMs, as a type of mixture model, have enough degree of freedom (number of hidden states) and thus, are capable of modeling a large class of distributions. By AR-HMM one can capture both the long and short range dependencies, as it combines a Markov chain on the hidden variables, and statistical dependencies on the observed variables (Berchtold 1999). An HMM is a statistical model which is an extension of Markov chains (Baum and Petrie 1966). Such models are widely used in econometrics (Hamilton 1990), especially for predictions of time series (Fraser 2008; Sipos and Levendovszky 2013). Thus, due to the general capability of AR-HMM in capturing the stochastic characteristics of a wider range of random processes, we use AR-HMM modeling of the portfolio price series. In this way, the method proposed in this paper can also be applied even when the underlying VAR(1) assumption (necessary for OU modeling) does not hold.

In AR-HMM modeling, the current state is not directly visible to the observer, but each state emits an observable output quantity denoted by $\mathbf{X} = \{p_1, p_2, ..., p_T\}$. Unlike the standard HMM assumption, in the case of AR-HMMs the emissions are conditionally not independent given the hidden state $\mathbf{Q} = \{q_1, q_2, ..., q_T\}$ (Murphy 2012). In our approach, the observable output was treated as a continuous value, described by a Gaussian probability density function. Then the probability of emitting a specific output is determined by the conditional probability

$$P\left(p\left(t\right)|p\left(t-1\right),\ q_t = j, \Theta\right) = N\left(p\left(t\right)\left|\varphi_j p\left(t-1\right) + \mu_j, \sigma_j\right.\right) \tag{2.3}$$

In other words, the observation depends on the hidden state, and on the previous observation through an additive autoregressive component. (Note that the μ and σ symbols were already introduced in (2.2), however, in this case they are vector quantities.) The transition probabilities of the underlying Markov chain, describing the jumping probabilities from one state to another is given by the transition probability matrix $A_{ij} = P\left(q(t+1) = q_i\left|q(t) = q_j\right.\right)$, while $\boldsymbol{\pi}$ denotes the initial distribution vector. The complete model then described by

$$\Theta = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\varphi}, \boldsymbol{\mu}, \boldsymbol{\sigma}\} \tag{2.4}$$

The generality of AR-HMM stems form the fact that we can consider a single state AR-HMM being equivalent as an OU process (Sipos and Levendovszky 2015), however, this can be either a mean reverting or a mean averting process, and a single state AR-HMM state can represent Brownian Motion (BM) with a possible linear drift as well. Furthermore, having multiple states brings more flexibility into modeling the price distributions, and this model can also capture if the process is driven by different stochastic rules over different intervals in time [often called as regimes (Hamilton and Susmel 1994)].

3 Portfolio Optimization

In this section we discuss the optimal portfolio selection subject to a new objective function. First, let us define the lower bound for the future price of a given portfolio achieved with a predefined probability η, i.e.:

$$\omega(t) : P\left(p(t) \geq \omega(t)\right) = P\left(\mathbf{x}^T \mathbf{s}_t \geq \omega(t)\right) = \eta \tag{3.1}$$

As a result, $\omega(t)$ is determined by the complementary cumulative probability distribution of the portfolio value, as a random variable. Then the objective function maximizes the return based on the lower bound as follows (see also Fig. 1):

$$\mathbf{x}_{opt} = \underset{\mathbf{x}}{\operatorname{argmax}}\ \Psi(\mathbf{x}), card(\mathbf{x}) \leq l, \tag{3.2}$$

where

$$\begin{aligned} \Psi(\mathbf{x}) = \max_{0<t} \omega(t) - p(0) \\ \text{under the constraint that } E\left(p(t)\right) - p(0) > 0. \end{aligned} \tag{3.3}$$

Taking into account the cardinality constraint as well, portfolio optimization can be reduced to a constrained optimization problem. However, attention should be given when the future portfolio price distribution is asymmetric. Especially, in the case when its expected value implies loss on the portfolio, while at the same time the median ($\eta = 0.5$) would falsely prompt us to invest into this specific portfolio. In order to avoid such situations, a constraint is formed accordingly.

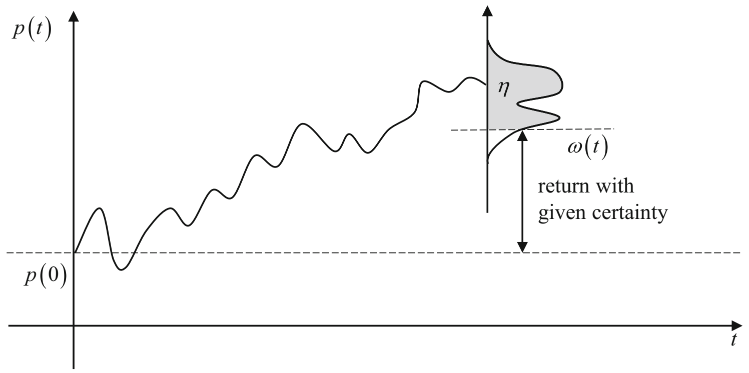

Fig. 1 The objective function maximizes the return based on the tail distribution

This objective function may prove to be much more beneficial for the portfolio holder than just simply optimizing the predictability, however, its solution does not lend itself to analytical tractability [(as opposed to the maximal predictability which leads to a generalized eigenvalue problem (D'Aspremont 2011)]. Thus, to come to grip with maximizing $\omega(t)$ for a given portfolio, we identify AR-HMM model parameters for an observed process, and then, as a novel approach, we calculate the prediction based complementary cumulative distribution function (CCDF).

To obtain the CCDF of the future values of $p(t)$, we have used Monte Carlo method. K simulations were run on the identified AR-HMM starting from the observed initial value of $p(0) = \mathbf{x}^T \mathbf{s}_0$ in the $0 < t \leq T_{\max}$ interval. Section 4.2 describes methods for estimating $\Theta = \{\boldsymbol{\pi}, \mathbf{A}, \boldsymbol{\varphi}, \boldsymbol{\mu}, \boldsymbol{\sigma}\}$ and $\boldsymbol{\gamma}_0$. Parameter η allows the investors to adjust their willingness to take risk. One can see that $\eta < 0.5$ results in a braver strategy, while $\eta > 0.5$ yields a rather cautious one.

One can include the bid-ask spread into this model in a straightforward manner, i.e. the portfolio prices, on which the AR-HMM is fit, should then be calculated as $p(t) = \mathbf{x}^T_{(long)}\mathbf{s}^{bid}_t + \mathbf{x}^T_{(short)}\mathbf{s}^{ask}_t$, while the current price in the objective function as $p(0) = \mathbf{x}^T_{(long)}\mathbf{s}^{ask}_0 + \mathbf{x}^T_{(short)}\mathbf{s}^{bid}_0$, where $x_i^{(long)} = \begin{cases} x_i, & if\ x_i > 0 \\ 0, & if\ x_i \leq 0 \end{cases}$ and similarly $x_i^{(short)} = \begin{cases} x_i, & if\ x_i < 0 \\ 0, & if\ x_i \geq 0 \end{cases}$. Negative values in the portfolio vector means short selling the assets.

4 Computational Approach to Modeling and Optimization

The AR-HMM model parameter identification and the evaluation of the objective function (3.3) is carried out driven by simulated annealing (see 4.1), which helps us select an optimal portfolio. Then, the trading signal for taking the appropriate trading actions is made on the basis of the identified portfolio and the cost function. Our computational framework is shown by the following block diagram (Fig. 2) and detailed as follows:

- Generate a portfolio vector under the constraint driven by Simulated Annealing (SA);

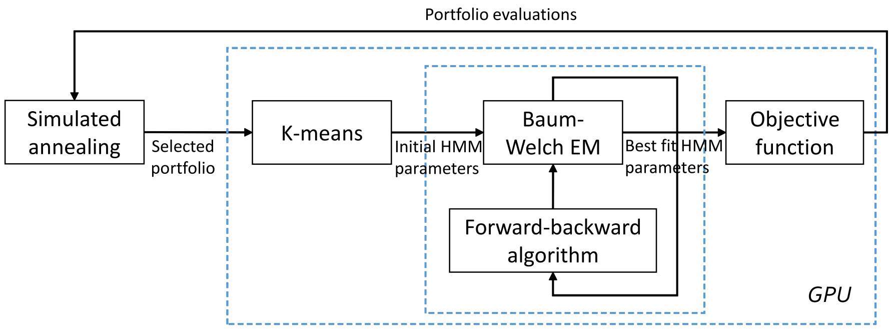

Fig. 2 Computational approach

- Fit an AR-HMM to the time series of the selected portfolio by using the Baum–Welch learning algorithm described in Sect. 4.2;
- Evaluate the objective function by predicting the future asset prices according to the identified model given the portfolio (see Sect. 3);
- Continue the simulated annealing process until the optimal portfolio is obtained (as detailed in Sect. 4.1) according to the objective function;
- Form a trading signal based on the price behavior of the optimal portfolio (Sect. 4.3) to decide on which trading action is to be launched;

Finally, one can carry out a performance analysis by testing and evaluating various numerical indicators for the sake of comparing the profitability of the different methods (Sect. 6 contains further details).

4.1 Simulated Annealing for Portfolio Optimization

This section focuses on the portfolio optimization. In the absence of proper analytical solutions for the constrained optimization problem posed in (3.2), we use simulated annealing to obtain good quality (as close to the global optimum as possible) heuristic solutions (Fogarasi and Levendovszky 2013).

Simulated annealing (SA) is a stochastic search method for finding the global optimum in a large search space as described in many related papers (Fogarasi and Levendovszky 2013; Janaki et al. 1996; Kirkpatrick et al. 1983; Sipos and Levendovszky 2013). In our case the negative energy function $J(\mathbf{x})$ is equivalent with the objective function of $\omega(t)$ defined as follows:

$$-J(\mathbf{x}) = \Psi(\mathbf{x}) = \max_{0<t} \omega(t) - p(0) \tag{4.1}$$

We perform parallel and independent simulated annealing in a given number of randomly selected subspaces. Each subspace is l dimensional, to fulfill the cardinality constraint $card(\mathbf{x}) \leq l$. More precisely, we used closed orthants in $\mathbf{R}^l$, constraining each coordinate of the subspace to be nonnegative or nonpositive. This is motivated by our investigation which has shown that SA converges faster and provide more reliable results in these regions. However, an exhaustive search in every subspace is computationally not feasible in case of a larger number of assets. The neighbor function on each iteration generates a new portfolio on the L1-ball, the distance of which from the previous one depends on the current temperature T. The algorithm automatically ensures that the available cash is not exceeded either with the long positions, or the short positions. In each of the selected subspaces let $\mathbf{x}$ be an arbitrary initialization vector, and then by calling a random number generation a new vector $\mathbf{x}'$ is generated randomly subject to the abovementioned neighbor function. We automatically, accept the new vector if $J\left(\mathbf{x}'\right) < J(\mathbf{x})$, or if $J\left(\mathbf{x}'\right) \geq J(\mathbf{x})$ then due to random acceptance with $e^{-\frac{J\left(\mathbf{x}'\right)-J(\mathbf{x})}{T}}$ probability. The sampling is then continued while decreasing the T until zero. Finally, the last state vectors obtained in the corresponding subspaces are compared, and the one minimizing $J(\mathbf{x})$ is now the identified optimal sparse portfolio vector.

4.2 HMM Parameter Optimization

The estimation of the model parameters (learning) is a key aspect when we are using HMMs for prediction. First, we need to sample the portfolio price through a sliding window of length T:

$$\mathbf{X} = \{p_{i-T+1}, ..., p_i\} \tag{4.2}$$

During training, the likelihood (or in practice, due to the small order of magnitude of such probabilities, the log-likelihood) of the model is maximized based on the given observations (Rabiner and Juang 1986):

$$\Theta_{opt} = \arg\max_{\Theta} P\left(\mathbf{X}|\Theta\right) \tag{4.3}$$

The Baum–Welch expectation maximization (EM) algorithm (Baum 1970) is a mechanism to iteratively update the model (2.4) starting from an arbitrary initial value and iterating until the likelihood of the model converges to a certain value. Since this is an iterative method, which can use the forward-backward algorithm, implemented in an efficient way by dynamic programming (Rabiner 1989), this algorithm is relatively fast. On the other hand, it may get stuck in one of the local minima. Compared to the standard HMM, in the case of an AR-HMM, the forward-backward algorithm remains unchanged, while in the EM algorithm only a slight modification needed (Dey et al. 1994). We can also obtain the posterior state probabilities from this procedure, the conditional probabilities of being at state j at time instance t given the observation sequence:

$$\gamma_t{}^{(j)} = P\left(q_t = j\,|\mathbf{X}\right) \tag{4.4}$$

4.3 Trading Strategy

In this section we describe the trading algorithm which is used to trade with the selected optimal portfolio based on the evaluation of the objective function (Sipos and Levendovszky 2013). In the proposed algorithm, trading is described as a "walk" in a binary state space in which either we already have a portfolio at hand or cash at hand, while the transitions are only affected by the evaluations of the potentially owned and the newly identified portfolios subject to the objective function (see Sect. 3). The trading strategy is depicted by a state chart (Fig. 3).

As of (3.3), positive evaluation indicates a profitable portfolio, while negative evaluation indicates that the portfolio may produce a loss. Higher value in each objective function implies a better portfolio. Based on this, the agent buys a portfolio only if it has a positive evaluation. A new trading action is taken if a newly identified portfolio ($\mathbf{x}_{opt}$) has higher and also positive evaluation than the present one ($\mathbf{x}$). In this case one can sell the owned portfolio and buy the new one with higher expectations instead. This approach treats the present portfolio as a sunk cost, thus only the future expectations are taken into consideration. Hence, we do not have to give up the best available portfolio in favor of a presently unfavorable portfolio. In the case that neither

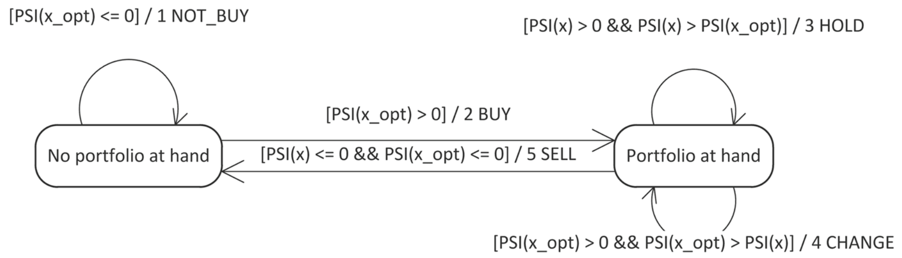
[PSI(x_opt) <= 0] / 1 NOT_BUY
[PSI(x) > 0 && PSI(x) > PSI(x_opt)] / 3 HOLD
[PSI(x_opt) > 0] / 2 BUY
No portfolio at hand
[PSI(x) <= 0 && PSI(x_opt) <= 0] / 5 SELL
Portfolio at hand
[PSI(x_opt) > 0 && PSI(x_opt) > PSI(x)] / 4 CHANGE

Fig. 3 Trading strategy

the owned nor the currently identified portfolio has positive evaluation then the agent closes the position (Fig. 3).

5 Parallel Implementation

In this section, we investigate the implementation of AR-HMM based portfolio optimization on parallel architectures. This implementation may pave the way towards a more sophisticated HFT using AR-HMMs on GPGPU. Existing solutions [(Idvall and Jonsson (2008) or Tenyakov and Mamon (2013)] can handle only a single HMM with low number of assets, whereas our solution enables to accommodate several AR-HMMs in a real-time manner. This allows us to model portfolio prices instead of the asset prices, providing a better grasp on the number of free parameters. Having the option to fit multiple AR-HMMs in a parallel way also enables us to find a better quality HMM during the learning phase. However, there are some algorithmic challenges to be resolved when implementing the portfolio optimization on multicore architectures.

The architecture which can give rise to parallel implementation is the GPU. It is a many-core processor with tremendous computational power and very high memory bandwidth (NVIDIA 2014). The reason behind the discrepancy in floating-point capability between the CPU and the GPU is that GPU is designed for compute-intensive, parallel computation. Compared to the simple CPU containing only few cores, a modern graphics processor has multiple thousand cores (NVIDIA TESLA K40—2880 cores).

A typical GPU consists two key components. The first is the array of Streaming Multiprocessors (SM), which contains many computation cores and some on-chip memory. The on-chip—64 kilobytes—memory is shared between processing cores in one SM. The second important part is the device memory that is much larger than the on-chip memory −1 to 6 gigabytes commercialized till 2014—and shared between each SM (Lindholm et al. 2008). However this global memory is very large compared to the on-chip memory and it is two magnitude slower. Therefore, to use the GPU optimally we minimize memory transfers from global memory to on-chip and vice versa.

We use parallel simulated annealing to obtain the optimal portfolio as described in Sect. 4.1. Random initial portfolios are generated in each subspace (satisfying the cardinality constraint), and for each portfolio we fit an HMM and evaluate the objective function (3.3). The HMM model parameter identification and the HMM based prediction is fully implemented on the GPU. Then we let the parallel optimization algorithm run until reaching the predefined iteration number, and then we pick the best portfolio with respect to the objective function.

In order to run the Baum–Welch algorithm for identifying the HMM parameters we need an initial estimation. The initial parameter estimations of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ in (2.4) are based on the K-means clustering algorithm. The input of K-means algorithm consists the time series for each portfolios and the outputs are the cluster centres and variances. For a given portfolio we can evaluate the clusterization multiple times (each started with different initial centers) which might converge into different state distributions. If we run the Baum–Welch algorithm from multiple starting points, we select the one HMM which likelihood is maximal. As the portfolios are independent from each other, we could increase the level of parallelism by running several K-means on the device parallel. This algorithm is used only to create a reasonable starting point for local optimization, and the data points are simply treated as samples from the equilibrium distribution, hence their order is not taken into consideration.

After identifying the initial state distribution, we use the Baum–Welch algorithm (as described in Sect. 4.2) to estimate the final states, together with the autoregressive $\boldsymbol{\varphi}$-s, and transition matrix per HMM. Training HMMs by the traditional Baum–Welch algorithm on a single core architecture is straightforward. It uses an Expectation-Maximization algorithm to find the model parameters for a given set of observation. The computationally most time consuming step of this process is the forward-backward algorithm.

In the parallel implementation, the number of parallel threads are the same as the number of states, on the other hand, updating the probability distribution vector is still to be done sequentially. As a result, the degree of parallelism is only determined by the number of states for a given HMM. This can be further extended by running several HMMs for each portfolio simultaneously (together with the K-means algorithm). In this way, the Baum–Welch algorithm can be performed independently on N number of threads. However, the number of states is chosen less than 32 because a further increase will not considerably improve the trading performance.

The conventional implementation of Baum–Welch and K-means algorithm run until they meet a stopping criteria (e.g. the objective function falls below a given threshold). Due to the nature of parallel computing we use a fixed number of iteration, because the runtime is determined by the slowest thread. Also, the sequential steps of the algorithms are also run on the GPU to avoid transferring data between the GPU and CPU causing long delays in each iteration.

Regarding the portfolio optimization, we perform parallel and independent simulated annealing in a given number of randomly selected subspaces, fulfilling the cardinality constraint $card(\mathbf{x}) \leq l$. This approach further increases the level of parallelism, resulting N threads per subspace in total. Depending on the properties of the actual GPU used for the calculations a so called block can be formed from one or more of these threads to be scheduled to the multiprocessors.

The parallelism of the proposed method is twofold with respect to granularity:

- *Coarse-grained parallelism* as opposed to the sequential SA in which in each run we start from a given initial condition, in the case of our parallel implemented SA several algorithms run from several initial points at the same time, i.e. AR-HMM parameter fittings in each subspace are done simultaneously.
- *Fine-grained parallelism* in each step of SA, (i) the algorithm for AR-HMM parameter estimation is parallel along the hidden states, and (ii) the evaluation of the objective function through the Monte Carlo simulations for estimating the complementary cumulative distribution function is also carried out in a parallel manner.

6 Performance Analysis

An extensive back-testing framework was created to handle trading actions on various input data sets and provide numerical results for the sake of comparing the performance of CPU and the GPGPU. For a detailed comparative analysis the following performance measures were calculated for each experiments on the corresponding time series (either FOREX or generated):

(i) Minimal value $G_{\min} = \frac{1}{c_0} \min_{0 \le t \le T} c_t$;
(ii) Final value $G_{final} = \frac{c_T}{c_0}$;
(iii) Maximal value $G_{\max} = \frac{1}{c_0} \max_{0 \le t \le T} c_t$;
(iv) Average value $G_{avg} = \frac{1}{c_0} \frac{1}{T} \sum_{t=1}^{T} c_t$,

where c_t denotes the sum of owned cash and the market value of the owned portfolio at time instance t, while c_0 denotes the initial cash (in each case the agent started with \$10,000).

In this section, we show the numerical results obtained on the following data sets:

- AR-HMM generated time series;
- FOREX rates (EUR/USD, GBP/USD, AUD/USD, NZD/USD, USD/CHF, USD/CAD from the year of 2013 in daily and 15 min resolution) [23].

The artificial data was generated by an AR-HMM with two and three hidden states, respectively. Regarding the sparsity constraint, three assets were selected in each transaction.

Each numerical experiments were running during the same time interval on both architectures. As the GPGPU is much faster and thus capable of visiting much more portfolios in the course of the optimization a much better value of the objective function has been achieved than the one calculated by CPU. Consequently, the trading performance has also surpassed the one achieved by the CPU (trading results on generated data is shown by Fig. 4). In each case 3 hidden states were used, we used 65 observations from the past to fit AR-HMMs, while the prediction was projected to the next 8 time instances.

The next figures indicate real tests running on FOREX data obtained at 15 min (Fig. 6) and daily (Fig. 5) tick time. The duration of tests have been 1 year and 3 days, respectively.

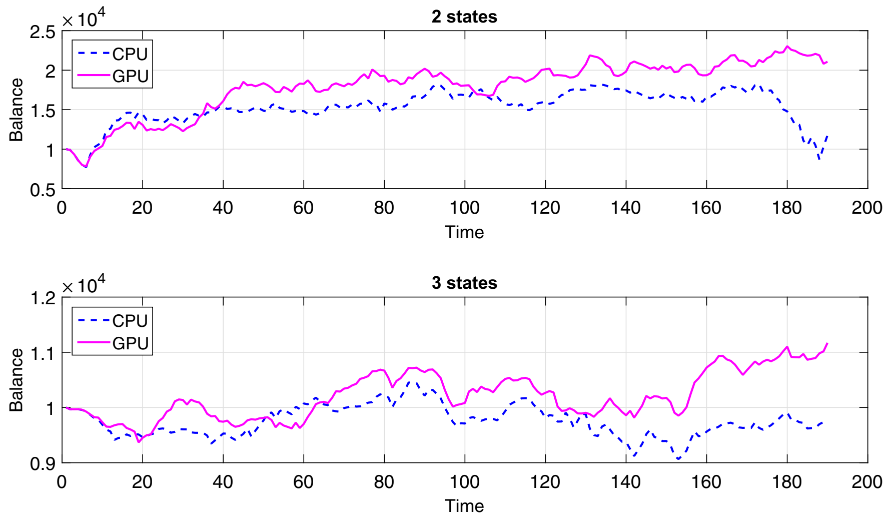

Fig. 4 Trading performance on generated data

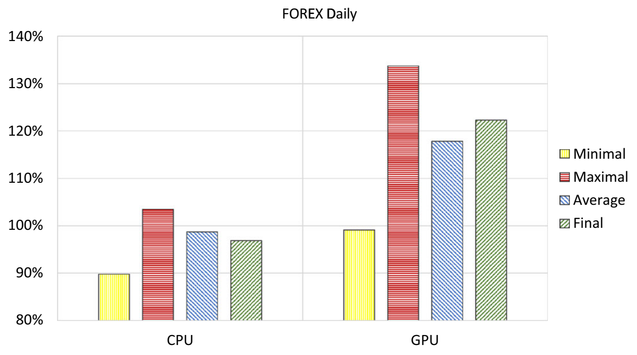

Fig. 5 Trading performance on daily FOREX data

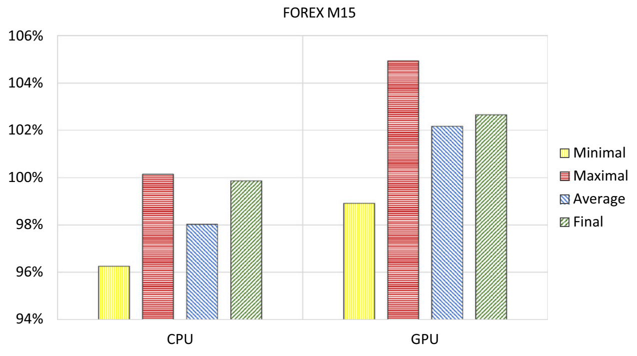

Fig. 6 Trading performance on 15-min FOREX data

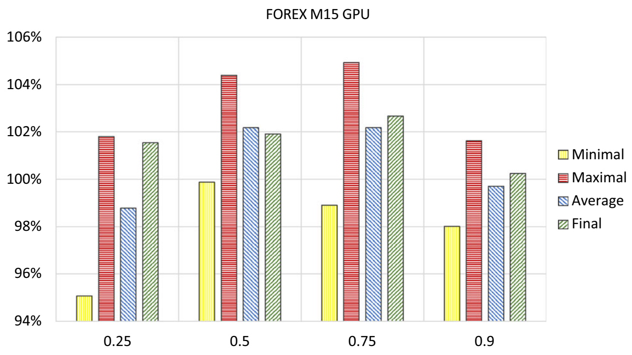

Fig. 7 Comparison of trading results using different η on 15-min FOREX data

One can see, in each case we managed to secure positive profit. On the 15 minute tick time the GPGPU has achieved 2.7 % profit, while 0.1 % loss achieved by CPU in a 3 days interval (annualization is often misleading on high-frequency data). On the daily averages the system performed more modestly, but even in this case the GPGPU secured a 22.3 % yearly profit as opposed to the 3.1 % loss achieved by the CPU. The reason for the poorer performance on daily averages may lie in the fact that in the case of more infrequent samples one cannot exploit the advantage of nature of the time series exhibited at a finer scale.

As demonstrated by Fig. 7, we also compared the performance when using different η parameters in (3.1), which controls the level of risk taking. As one can see, it proved to be favorable to follow a cautious strategy with $\eta = 0.75$, but not an overly cautious one.

6.1 Speed Profiling

The experiments were conducted on a PC with a second generation Intel i7 CPU and on one NVIDIA GeForce GTX 570 GPU. The comparison between the computation time of CPU and GPU is shown by Fig. 8.

Figure 9 shows the speedup over serial implementation, that is the ratio between the computation time on CPU and the computation time on GPGPU.

The performance in the case of small number states of HMM is relatively low, which is not surprising in the light of the fact the GPGPU computing power remains underutilized. If we increase the number of optimized HMMs then more and more blocks will become active on the GPGPU which, in turn, will increase the performance. Thus, the GPU will be much faster than the CPU with a factor 1024 up to 1270. This increase in speed is imperative to perform HFT. To demonstrate the difference between CPU and GPGPU, in the M15 simulations (as shown in Fig. 8) the portfolio optimization took 478 ms on average, which means about 10 minutes on CPU casting it unfeasible to use in this time scale.

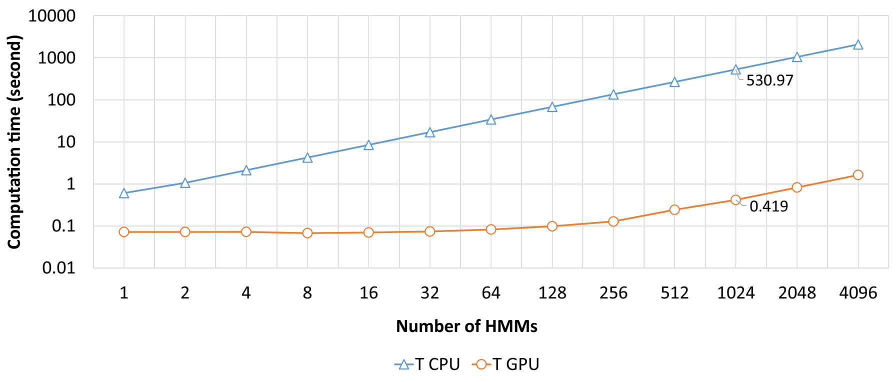

Fig. 8 Comparison between the computation time of CPU and GPU

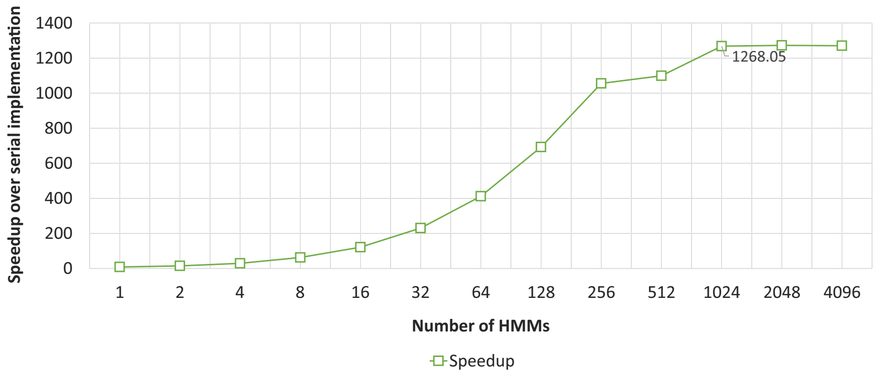

Fig. 9 Speedup over serial implementation

7 Conclusions

In this paper we have proposed novel algorithms for OU parameter estimation and predicting future values of financial time series with AR-HMMs for portfolio optimization subject to cardinality constraints. The portfolio optimizations have been carried out by stochastic search with respect to new objective function in which we maximize the return achieved by a given probability. The underlying computational processes have been implemented on GPGPU.

The proposed trading algorithms have proven to be profitable on real financial time series. The performance analysis demonstrated that the prediction (based on the novel parameter identification) could indeed increase the trading efficiency and the profit compared to the traditional OU modeling, where the parameters are estimated by linear regression. However, there is still a room for further improvement on the trading strategy in order to take other factors (e.g. to avoid early sells) into consideration, as well.

The implementation on GPGPU have proven that the GPGPU architecture can cope with high frequency trading and yields a superior performance to that of the CPU both in speed and in a selecting more profitable portfolios.

Acknowledgments This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (Grant No.: TAMOP-4.2.2.C-11/1/KONV-2012-0013). Also, we gratefully acknowledge the support of NVIDIA Corporation.

References

Baum, L. E., & Petrie, T. (1966). Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*, *37*(6), 1554–1563.

Baum, L. E., et al. (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *The Annals of Mathematical Statistics*, *41*(1), 164–171.

Becker, S., Cevher, V., Koch, C., & Kyrillidis, A. (2013). Sparse projections onto the simplex. ICML (to appear).

Berchtold, A. (1999). The double chain Markov model. *Communications in Statistics-Theory and Methods*, *28*(11), 2569–2589.

Chang, T. J., Meade, N., Beasley, J. E., & Sharaiha, Y. M. (2000). Heuristics for cardinality constrained portfolio optimisation. *Computers & Operations Research*, *27*(13), 1271–1302.

Cook, S. (2013). *CUDA programming: A developer's guide to parallel computing with GPUs*. Amsterdam: Elsevier.

D'Aspremont, A. (2011). Identifying small mean-reverting portfolios. *Quantitative Finance*, *11*(3), 351–364.

Dey, S., Krishnamurthy, V., & Salmon-Legagneur, T. (1994). Estimation of Markov-modulated time-series via EM algorithm. *Signal Processing Letters, IEEE*, *1*(10), 153–155.

Durbin, R. (Ed.). (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*. Cambridge: Cambridge University Press.

Fama, E., & French, K. (1988). Permanent and temporary components of stock prices. *The Journal of Political Economy*, *96*(2), 246–273.

Fieldsend, J. E., Matatko, J., & Peng, M. (2004) Cardinality constrained portfolio optimisation. In Intelligent data engineering and automated learning–IDEAL 2004 (pp. 788–793). Berlin: Springer.

Fogarasi, N., & Levendovszky, J. (2013). A simplified approach to parameter estimation and selection of sparse, mean reverting portfolios. *Periodica Polytechnica Electrical Engineering and Computer Science*, *56*(1), 21–28.

Fogarasi, N., & Levendovszky, J. (2013). Sparse, mean reverting portfolio selection using simulated annealing. *Algorithmic Finance*, *2*(3), 197–211.

Fraser, A. M. (2008). *Hidden Markov models and dynamical systems*. Philadelphia: SIAM.

Hamilton, J. D. (1990). Analysis of time series subject to changes in regime. *Journal of Econometrics*, *45*(1), 39–70.

Hamilton, J. D., & Susmel, R. (1994). Autoregressive conditional heteroskedasticity and changes in regime. *Journal of Econometrics*, *64*(1), 307–333.

Hassan, M. R., & Nath, B. (2005). Stock market forecasting using hidden Markov model: A new approach. In: *Intelligent systems design and applications, 2005. ISDA'05. Proceedings. 5th international conference on IEEE* (pp. 192–196).

Idvall, P., & Jonsson, C. (2008). Algorithmic trading: Hidden markov models on foreign exchange data. Master's thesis, Linköping University, Department of Mathematics.

Ito, K. (1944). Stochastic integral. *Proceedings of the Imperial Academy Tokyo*, *20*, 519–524.

Janaki, R. D., Sreenivas, T. H., & Ganapathy, K. S. (1996). Parallel simulated annealing algorithms. *Journal of Parallel and Distributed Computing*, *37*, 207–212.

Jurafsky, D., & Martin, J. H. (2006). *Speech and language processing: An introduction to natural language processing. Computational linguistics, and speech recognition* (2nd ed.). London: Pearson Prentice Hall.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680.

Lindholm, E., Nickolls, J., Oberman, S., & Montrym, J. (2008). NVIDIA Tesla: A unified graphics and computing architecture. *Micro IEEE*, *28*(2), 39,55.

Manzan, S. (2007). Nonlinear mean reversion in stock prices. *Quantitative and Qualitative Analysis in Social Sciences*, *1*(3), 1–20.

Mamon, R. S., & Elliott, R. J. (2007). *Hidden markov models in finance*. New York: Springer.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, *7*(1), 77–91.

Metatrader (2014) *FOREX time series* [WWW]. Accessed 02 2014, Available from, http://www.metaquotes.net/en/metatrader5.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. Cambridge: MIT Press.

Natarajan, B. K. (1995). Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, *24*(2), 227–234.

NVIDIA. (2014). *CUDA C programming guide* [WWW]. Accessed 08 2014, Available from, http://docs.nvidia.com/cuda/cuda-c-programming-guide.

Ornstein, L. S., & Uhlenbeck, G. E. (1930). On the theory of the Brownian motion. *Physical Review*, *36*(5), 823.

Pagès, G., & Wilbertz, B. (2011). GPGPUs in the computational finance: Massive parallel American style options. *Concurrency and Computation: Practice and Experience*, *24*, 837–848.

Poterba, J. M., & Summers, L. H. (1988). Mean reversion in stock prices: Evidence and implications. *Journal of Financial Economics*, *22*(1), 27–59.

Rabiner, L., & Juang, B.-H. (1986). An introduction to hidden Markov models. *ASSP Magazine, IEEE*, *3*(1), 4–16.

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257–286.

Shaw, D. X., Liu, S., & Kopman, L. (2008). Lagrangian relaxation procedure for cardinality-constrained portfolio optimization. *Optimisation Methods & Software*, *23*(3), 411–420.

Sipos, I. R., & Levendovszky, J. (2013). Optimizing sparse mean reverting portfolios. *Algorithmic Finance*, *2*(2), 127–139.

Sipos, I. Róbert, & Levendovszky, János. (2015). Optimizing sparse mean reverting portfolios with AR-HMMs in the presence of secondary effects. *Periodica Polytechnica Electrical Engineering and Computer Science*, *59*(1), 1–8.

Sipos, I. R., & Levendovszky, J. (2013). High frequency trading with Hidden Markov Models by using clustering and PPCA algorithms. In: *Proceedings, 15th applied stochastic models and data analysis (ASMDA2013)*, Barcelona.

Tenyakov, A., & Mamon, R. (2013). *Modelling high-frequency FX rate dynamics: A zero-delay multi-dimensional HMM-based approach*. Department of Statistical and Actuarial Sciences, University of Western Ontario.