

## Accelerated fluctuation analysis by graphic cards and complex pattern formation in financial markets\*

Tobias Preis<sup>1,2,3</sup>, Peter Virnau<sup>1</sup>, Wolfgang Paul<sup>1</sup>  
and Johannes J Schneider<sup>1</sup>

<sup>1</sup> Institute of Physics, Johannes Gutenberg University Mainz,  
Staudinger Weg 7, 55128 Mainz, Germany

<sup>2</sup> Artemis Capital Asset Management GmbH, Gartenstrasse 14,  
65558 Holzheim, Germany

E-mail: [preis@uni-mainz.de](mailto:preis@uni-mainz.de)

*New Journal of Physics* **11** (2009) 093024 (21pp)

Received 2 June 2009

Published 16 September 2009

Online at <http://www.njp.org/>

doi:10.1088/1367-2630/11/9/093024

**Abstract.** The compute unified device architecture is an almost conventional programming approach for managing computations on a graphics processing unit (GPU) as a data-parallel computing device. With a maximum number of 240 cores in combination with a high memory bandwidth, a recent GPU offers resources for computational physics. We apply this technology to methods of fluctuation analysis, which includes determination of the scaling behavior of a stochastic process and the equilibrium autocorrelation function. Additionally, the recently introduced pattern formation conformity (Preis *T et al* 2008 *Europhys. Lett.* **82** 68005), which quantifies pattern-based complex short-time correlations of a time series, is calculated on a GPU and analyzed in detail. Results are obtained up to 84 times faster than on a current central processing unit core. When we apply this method to high-frequency time series of the German BUND future, we find significant pattern-based correlations on short time scales. Furthermore, an anti-persistent behavior can be found on short time scales. Additionally, we compare the recent GPU generation, which provides a theoretical peak performance of up to roughly  $10^{12}$  floating point operations per second with the previous one.

\* Supplementary information can be found on <http://www.tobiaspreis.de>

<sup>3</sup> Author to whom any correspondence should be addressed.

**Contents**

<b>1. Introduction</b>	<b>2</b>
<b>2. GPU device architecture</b>	<b>4</b>
<b>3. Hurst exponent</b>	<b>6</b>
<b>4. Equilibrium autocorrelation</b>	<b>10</b>
<b>5. Fluctuation pattern conformity</b>	<b>13</b>
<b>6. Conclusion and outlook</b>	<b>18</b>
<b>Acknowledgments</b>	<b>18</b>
<b>Appendix. GPU source code</b>	<b>19</b>
<b>References</b>	<b>19</b>

**1. Introduction**

In computer science applications and diverse interdisciplinary science fields such as computational physics or quantitative finance, the computational power requirements increase monotonically in time. In particular, the history of time series analysis mirrors the needs of computational power and simultaneously the opportunities arising from the use of it. Up to the present day, it is an often made simplistic assumption that price dynamics in financial time series obey random walk statistics in order to simplify analytic calculations in econophysics and in financial applications. However, such approximations, which are, e.g. used in the famous options pricing model introduced by Black and Scholes [1] in 1973, neglect the real nature of financial market observables, and a large number of empirical deviations between financial market time series and models presuming only a random walk behavior have been observed in recent decades [2]–[6]. Already Mandelbrot [7, 8] discovered in the 1960s that commodity market time series obey fat-tailed price change distributions [9]. His analysis was based on historical cotton times and sales records dating back to the beginning of the 20th century. In accordance with the technological improvements in computing resources, trading processes were adapted in order to create full-electronic market places. Thus, the available amount of historical price data increased impressively. As a consequence, the empirical properties found by Mandelbrot were confirmed. However, the amount of transaction records available today in time units of milliseconds also requires increased computing resources for its analysis. From such analyses, scaling behavior, short-time anti-correlated price changes and volatility clustering [10, 11] of financial markets are well established and can be reproduced, e.g. by a model of the continuous double auction [12, 13] or by various agent-based models of financial markets [14]–[22]. Furthermore, the price formation process and cross correlations [23, 24] between equities and equity indices have been studied with the clear intention to optimize asset allocation and portfolios. However, in contrast to such calculations, which can be done with conventional computing facilities, a large computational power demand is driven by the quantitative hedge fund industry and also by modern market making, which requires mostly real time analytics. A market maker usually provides quotes for buying or selling a given asset. In the competitive environment of electronic financial markets this cannot be done by a human market maker alone, especially if a large number of assets is quoted concurrently. The rise

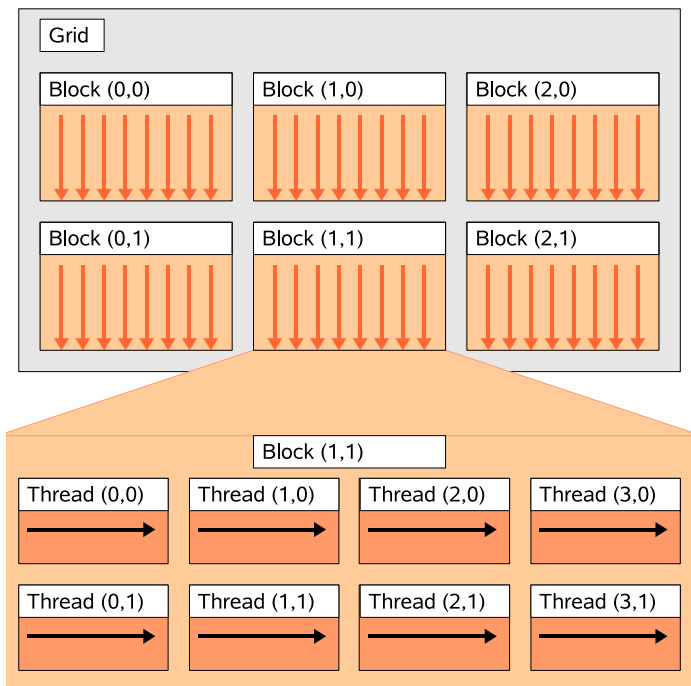
of the hedge fund industry in recent years and their interest in taking advantage of short time correlations boosted the real-time analysis of market fluctuations and the market micro-structure analysis in general, which is the study of the process of exchanging assets under explicit trading rules [25], and which is studied intensely by the financial community [26]–[29].

Such computing requirements, which can be found in various interdisciplinary computer sciences like computational physics including, e.g. Monte Carlo and molecular dynamics simulations [30]–[32] or stochastic optimization [33], make use of the high-performance computing resources necessary. This includes recent multi-core computing solutions based on a shared memory architecture, which are accessible by OpenMP [34] or MPI [35] and can be found in recent personal computers as a standard configuration. Furthermore, distributed computing clusters with homogeneous or heterogeneous node structures are available in order to parallelize a given algorithm by separating it into various sub-algorithms.

However, a recent trend in computer science and related fields is general purpose computing on graphics processing units (GPUs), which can yield impressive performance, i.e. the required processing times can be reduced to a great extent. Some applications have already been realized in computational physics [31], [36]–[43]. Recently, the Monte Carlo simulation of the two-dimensional and three-dimensional ferromagnetic Ising model could be accelerated up to 60 times [44] using a graphic card architecture. With multiple cores connected by high memory bandwidth, today's GPUs offer resources for non-graphics processing. In the beginning, GPU programs used C-like programming environments for kernel execution such as OpenGL shading language [45] or C for graphics (Cg) [46]. The compute unified device architecture (CUDA) [47] is an almost conventional programming approach making use of the unified shader design of recent GPUs from NVIDIA corporation. The programming interface allows one to implement an algorithm using standard C language without any knowledge of the native programming environments. A comparable concept 'Close To the Metal' (CTM) [48] was introduced by Advanced Micro Devices Inc. for ATI graphics cards. One has to state that computational power of consumer graphics cards roughly exceeds that of a central processing unit (CPU) by 1–2 orders of magnitudes. A conventional CPU nowadays provides a peak performance of roughly  $20 \times 10^9$  floating point operations per second (FLOPS) [31]. The consumer graphics card NVIDIA GeForce GTX 280 reaches a theoretical peak performance of  $933 \times 10^9$  FLOPS. If one tried to realize the computational power of one GPU with a cluster of several CPUs, a much larger amount of electric power would be required. A GTX 280 graphics card exhibits a maximum power consumption of 236 W [49], while a recent Intel CPU consumes roughly 100 W.

We apply this general-purpose graphics processing unit (GPGPU) technology to methods of time series analysis, which includes determination of the Hurst exponent and equilibrium autocorrelation function. Additionally, the recently introduced pattern conformity observable [50], which is able to quantify pattern-based complex short-time correlations of a time series, is calculated on a GPU. Furthermore, we compare the recent GPU generation with the previous one. All methods are applied to a high-frequency data set of the Euro-Bund futures contract traded at the electronic derivatives exchange Eurex.

The paper is organized as follows. In section 2, a brief overview of key facts and properties of the GPU architecture is provided in order to clarify implementation constraint details for the following sections. A GPU-accelerated Hurst exponent estimation can be found in section 3. In section 4, the equilibrium autocorrelation function is implemented on a GPU and in section 5, the pattern conformity is analyzed on a GPU in detail. In each of these sections, the performance



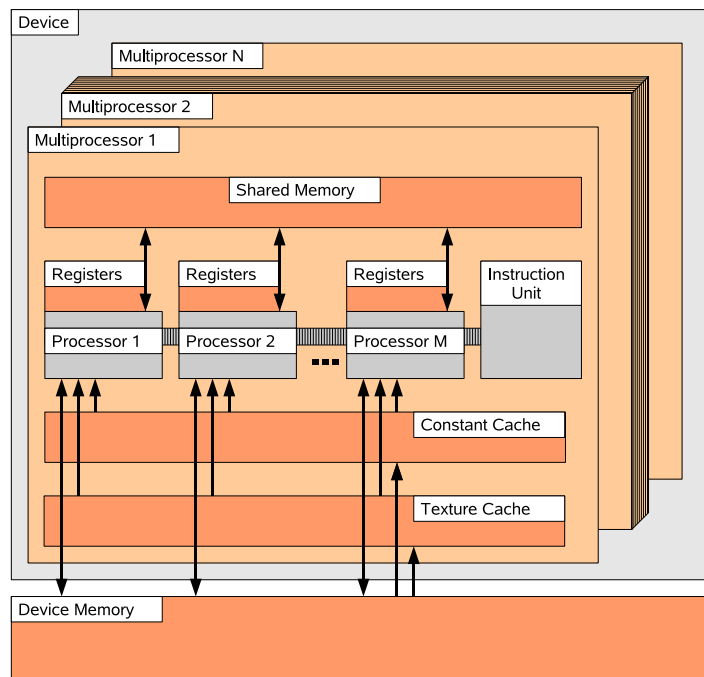
**Figure 1.** Schematic visualization of the grid of thread blocks for a two-dimensional thread and block structure.

of the GPU code as a function of parameters is first evaluated for a synthetic time series and compared to the performance on a CPU. Then the time series methods are applied to a financial market time series and a discussion of numerical errors is presented. Finally, our conclusions are summarized in section 6.

## 2. GPU device architecture

In order to provide and discuss information about implementation details on a GPU for time series analysis methods, key facts of the GPU device architecture are briefly summarized in this section. As mentioned in the introduction, we use the compute unified device architecture (CUDA), which allows the implementation of algorithms using standard C language with CUDA specific extensions. Thus, CUDA issues and manages computations on a GPU as a data-parallel computing device.

The graphics card architecture, which is used in recent GPU generations, is built around a scalable array of streaming MPs [47]. One such MP contains amongst others eight scalar processor cores, a multi-threaded instruction unit, and shared memory, which is located on-chip. When a C program using CUDA extensions and running on the CPU invokes a GPU kernel, which is a synonym for a GPU function, many copies of this kernel—known as threads—are enumerated and distributed to the available MPs, where their execution starts. For such an enumeration and distribution, a kernel grid is subdivided into blocks and each block is subdivided into various threads as illustrated in figure 1 for a two-dimensional thread and block structure. The threads of a thread block are executed concurrently in the vacated MPs.



**Figure 2.** Schematic visualization of GPU MPs with on-chip shared memory.

In order to manage a large number of threads, a single-instruction multiple-thread (SIMT) unit is used. An MP maps each thread to one scalar processor core and each scalar thread executes independently. Threads are created, managed, scheduled and executed by this SIMT unit in groups of 32 threads. Such a group of 32 threads forms a warp, which is executed on the same MP. If the threads of a given warp diverge via a data-induced conditional branch, each branch of the warp is executed serially and the processing time of this warp consists of the sum of the branches' processing times.

As shown in figure 2, each MP of the GPU device contains several local 32-bit registers per processor, memory that is shared by all scalar processor cores of an MP. Furthermore, constant and texture cache are available, which is also shared on an MP. In order to afford reducing results of involved MPs, slower global memory can be used, which is shared among all MPs and is also accessible by the C function running in the CPU. Note that the GPU's global memory is still roughly 10 times faster than the current main memory of personal computers. Detailed facts of the consumer graphics cards 8800 GT and GTX 280 used by us can be found in table 1. Furthermore note that a GPU device only supports single-precision floating-point operations, with the exception of the most modern graphic cards starting with the GTX 200 series. However, the IEEE-754 standard for single-precision numbers is not completely realized. Deviations can be found especially for rounding operations. In contrast, the GTX 200 series supports also double-precision floating-point numbers. However, each MP features only one double-precision processing core and so, the theoretical peak performance is dramatically reduced for double-precision operations. Further information about the GPU device properties and CUDA can be found in [47].

**Table 1.** Key facts and properties of the applied consumer graphics cards. The theoretical acceleration factor between GeForce 8800 GT and GeForce GTX 280 is given by the difference in number of cores  $\times$  clock rate.

	GeForce 8800 GT	GeForce GTX 280
Global memory	512 MB	1024 MB
Number of multiprocessors (MPs)	14	30
Number of cores	112	240
Constant memory	64 kB	64 kB
Shared memory per block	16 kB	16 kB
Registers available per block	8192	16384
Warp size	32	32
Clock rate	1.51 GHz	1.30 GHz

### 3. Hurst exponent

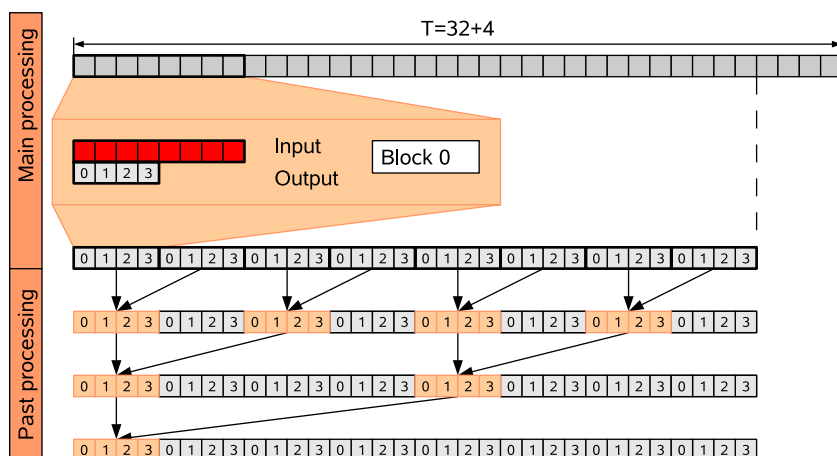
The Hurst exponent  $H$  [51] provides information on the relative tendency of a stochastic process. A Hurst exponent  $H < 0.5$  indicates an anti-persistent behavior of the analyzed process, which means that the process is dominated by a mean reversion tendency.  $H > 0.5$  mirrors a super-diffusive behavior of the underlying process. Extreme values tend to be followed by extremal values. If the deviations from the mean values of the time series are independent, which corresponds to a random walk behavior, a Hurst exponent of  $H = 0.5$  is obtained.

The Hurst exponent  $H$  was originally introduced by Harold Edwin Hurst [52], a British government administrator. He studied records of the Nile river's volatile rain and drought conditions and noticed interesting coherences for flood periods. Hurst observed in the eight centuries of records that there was a tendency for a year with good flood conditions to be followed by another year with good flood conditions. Nowadays, the Hurst exponent as a scaling exponent is well studied in context of financial markets [50], [53]–[56]. Typically, an anti-persistent behavior can be found on short timescales due to the nonzero gap between offer and demand. On medium timescales, a super-diffusive behavior can be detected [54]. On long timescales, a diffusive regime is reached, due to the law of large numbers.

For a time series  $p(t)$  with  $t \in \{1, 2, \dots, T\}$ , the time lag-dependent Hurst exponent  $H_q(\Delta t)$  can be determined by the general relationship

$$\langle |p(t + \Delta t) - p(t)|^q \rangle^{1/q} \propto \Delta t^{H_q(\Delta t)} \quad (1)$$

with the time lag  $\Delta t \ll T$  and  $\Delta t \in \mathbb{N}$ . The brackets  $\langle \dots \rangle$  denote the expectation value. Apart from (1), there are also other calculation methods, e.g. the rescaled range analysis [51]. We present in the following the Hurst exponent determination implementation on a GPU for  $q = 1$  and use  $H(\Delta t) \equiv H_{q=1}(\Delta t)$ . The process to be analyzed is a synthetic anti-correlated random walk, which was introduced in [50]. This process emerges from the superposition of two random walk processes with different timescale characteristics. Thus, a parameter-dependent anti-correlation at time lag one can be realized. As a first step, one has to allocate memory on the GPU device's global memory for the time series, intermediate results and final results. In a first approach, the time lag-dependent Hurst exponent is calculated up to  $\Delta t_{\max} = 256$ .

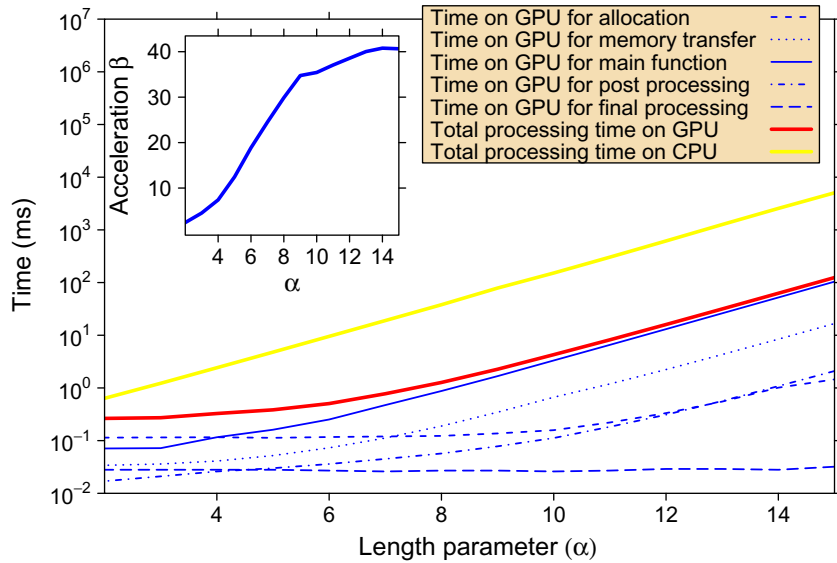


**Figure 3.** Schematic visualization of the determination of the Hurst exponent on a GPU architecture for  $T = 32 + 4$  and  $\Delta t_{\max} = 4$ . The calculation is split into various processing steps to ensure that their predecessors are completed.

In order to simplify the reduction process of the partial results, the overall number of time steps  $T$  has to satisfy the condition  $T = (2^\alpha + 1) \times \Delta t_{\max}$ , with  $\alpha$  being an integer number called the length parameter of the time series. The number of threads per block—known as block size—is equivalent to  $\Delta t_{\max}$ . The array for intermediate results possesses length  $T$  too, whereas the array for the final results contains  $\Delta t_{\max}$  entries. After allocation, the time series data have to be transferred from the main memory to the GPU's global memory. If this step is completed, the main calculation part can start. As illustrated in figure 3 for block 0, each block, which contains  $\Delta t_{\max}$  threads each, loads  $\Delta t_{\max}$  data points of the time series from global memory to shared memory. In order to realize such a high-performance loading process, each thread<sup>4</sup> loads one value and stores this value in the array located in the shared memory, which can be accessed by all threads of a block. Analogously, each block also loads the next  $\Delta t_{\max}$  entries. In the main processing step, each thread is in charge of one specific time lag. Thus, each thread is responsible for a specific value of  $\Delta t$  and summarizes the terms  $|p(t + \Delta t) - p(t)|$  in the block sub-segment of the time series. As the maximum time lag is equivalent to the maximum number of threads and as the maximum time lag is also equivalent to half the data points loaded per block, all threads have to summarize the same number of addends and so, a uniform workload of the graphics card is realized. However, as it is only possible to synchronize threads within a block and a native block synchronization does not exist, partial results of each block have to be stored in block-dependent areas of the array for intermediate results, as shown in figure 3. The termination of the GPU kernel function ensures that all blocks were executed. In a post processing step, the partial arrays have to be reduced. This is realized by a binary tree structure, as indicated in figure 3. After this reduction, the resulting values can be found in the first  $\Delta t_{\max}$  entries of the intermediate array and a final processing kernel is responsible for normalization and gradient calculation. The source code of these GPU kernel functions can be found in the appendix.

For the comparison between CPU and GPU implementations, we use an Intel Core 2 Quad CPU (Q6700) with 2.66 GHz and 4096 kB cache size, of which only one core is used. The

<sup>4</sup> Thread and block IDs are accessible in standard C language via built-in variables.



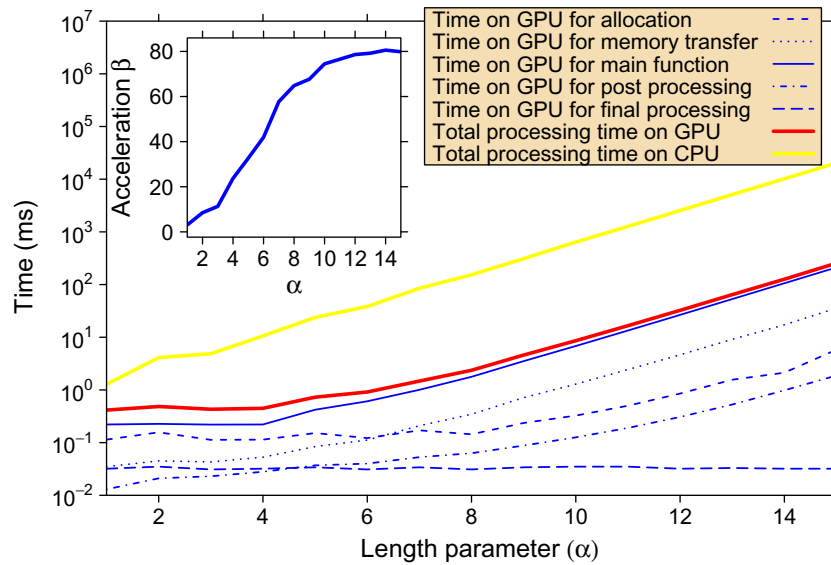
**Figure 4.** Processing times for the calculation of the Hurst exponent  $H(\Delta t)$  on GPU and CPU for  $\Delta t_{\max} = 256$ . The consumer graphics card 8800 GT is used as GPU device. The total processing time on the GPU can be broken into allocation time, time for memory transfer, time for main processing, time for post-processing, and time for final processing. The acceleration factor  $\beta$  is shown in the inset. A maximum acceleration factor of roughly 40 can be obtained. Furthermore, at  $\alpha = 9$  there is a break of the slope of the acceleration, which is influenced by cache size effects.

standard C source code executed on the host is compiled with the gcc compiler (version 4.2.1). The results for  $\Delta t_{\max} = 256$  and the consumer graphics card 8800 GT can be found in figure 4. The acceleration factor  $\beta$ , which is shown in the inset, reaches a maximum value of roughly 40, and is determined by the relationship

$$\beta = \frac{\text{Total processing time on CPU}}{\text{Total processing time on GPU}}. \quad (2)$$

A smaller speed-up factor can be measured for small values of  $\alpha$ , as the relative fraction of allocation time and time for memory transfer is larger in comparison to the time needed for the calculation steps. The corresponding analysis for the GTX 280 yields a larger acceleration factor  $\beta$  of roughly 70. If we increase the maximum time lag  $\Delta t_{\max}$  to 512, which is only possible for the GTX 280, a maximum speed-up factor of roughly 80 can be achieved, as shown in figure 5. This indicates that  $\Delta t_{\max} = 512$  leads to a higher workload on the GTX 280.

At this point, we can also compare the ratio between the performances of 8800 GT and GTX 280 for our application to the ratio of theoretical peak performances. The latter is given as the number of cores multiplied with the clock rate, which amounts to roughly 1.84. If we compare the total processing times on these GPUs for  $\alpha = 15$  and  $\Delta t_{\max} = 256$ , we obtain an empirical performance ratio of 1.7. If we use the acceleration factors for  $\Delta t_{\max} = 256$  on the 8800 GT and for  $\Delta t_{\max} = 512$  on the GTX 280 for comparison, we obtain a value of 2.

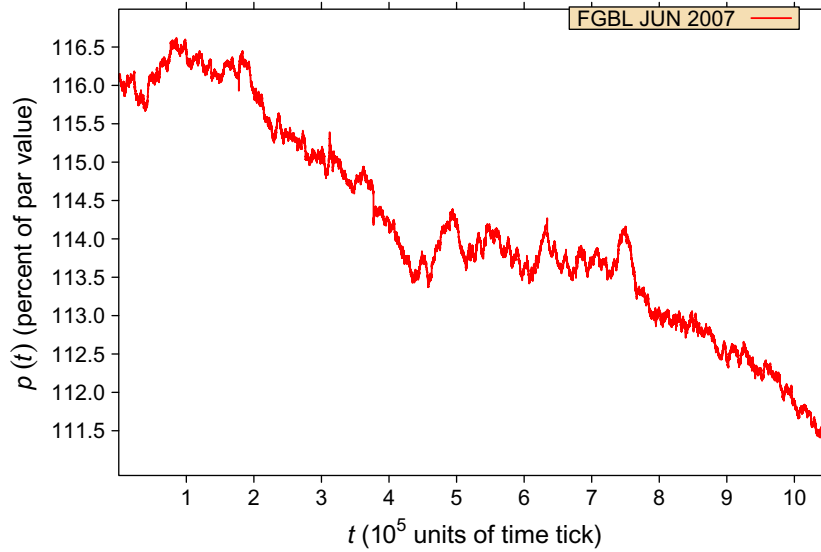


**Figure 5.** Processing times for the calculation of the Hurst exponent  $H(\Delta t)$  on GPU and CPU for  $\Delta t_{\max} = 512$ . These results are obtained by the GTX 280. The total processing time on GPU can also be broken into allocation time, time for memory transfer, time for main processing, time for post-processing and time for final processing. A maximum acceleration factor of roughly 80 can be reached, which is shown in the inset.

After this performance analysis, we apply the GPU implementation to real financial market data in order to determine the Hurst exponent of the Euro-Bund futures contract traded at the European exchange (Eurex). In this context, we will also gauge the accuracy of the GPU calculations by quantifying deviations from the calculation on a CPU. The Euro-Bund futures contract (FGBL) is a financial derivative. A futures contract is a standardized contract to buy or sell a specific underlying instrument at a proposed date in the future, which is called the expiration time of the futures contract, at a specified price. The underlying instruments of the FGBL contract are long-term debt instruments issued by the Federal Republic of Germany with remaining terms of 8.5 to 10.5 years and a coupon of 6%. We use the Euro-Bund futures contract with expiration time June 2007. The time series shown in figure 6 contains 1 051 982 trades, recorded from 8 March 2007 to 7 June 2007. In all presented calculations of the FGBL time series on the GPU,  $\alpha$  is fixed to 11. Thus, the data set is limited to the first  $T = 1\,049\,088$  trades in order to fit the data set length to the constraints of the specific GPU implementation. In figure 7, the time lag-dependent Hurst exponent  $H(\Delta t)$  is presented. On short timescales, the well-known anti-persistent behavior can be detected. On medium timescales, small evidence is given, that the price process reaches a super-diffusive regime. For long timescales the price dynamics tend to random walk behavior ( $H = 0.5$ ), which is also shown for comparison. The relative error

$$\epsilon = \left| \frac{H_{\text{GPU}}(\Delta t) - H_{\text{CPU}}(\Delta t)}{H_{\text{CPU}}(\Delta t)} \right| \quad (3)$$

shown in the inset of figure 7 is smaller than one-tenth of a per cent.



**Figure 6.** High-frequency financial time series of the FGBL with expiration time June 2007 traded at the Eurex: The time series contains 1 051 982 trades recorded from 8 March 2007 to 7 June 2007. In all presented calculations of the FGBL time series on a GPU,  $\alpha$  is fixed to 11. Thus, the data set is limited to the first  $T = 1\,049\,088$  trades in order to fit the data set length to the constraints of the specific GPU implementation.

#### 4. Equilibrium autocorrelation

The autocorrelation function is a widely used concept in order to determine dependencies within a time series. The autocorrelation function is given by the correlation between the time series and the time series shifted by the time lag  $\Delta t$  through

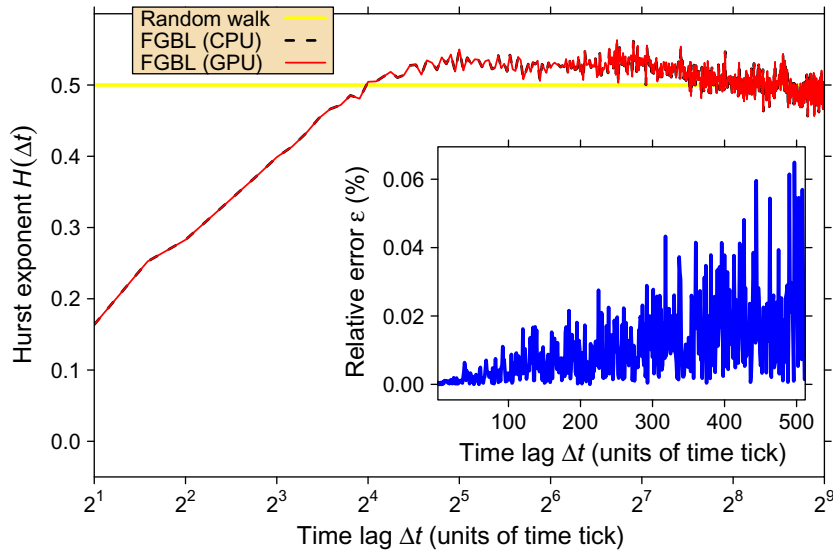
$$\rho(\Delta t) = \frac{\langle p(t) \cdot p(t + \Delta t) \rangle - \langle p(t) \rangle \langle p(t + \Delta t) \rangle}{\sqrt{\langle p(t)^2 \rangle - \langle p(t) \rangle^2} \sqrt{\langle p(t + \Delta t)^2 \rangle - \langle p(t + \Delta t) \rangle^2}}. \quad (4)$$

For a stationary time series, (4) reduces to

$$\rho(\Delta t) = \frac{\langle p(t) \cdot p(t + \Delta t) \rangle - \langle p(t) \rangle^2}{\langle p(t)^2 \rangle - \langle p(t) \rangle^2}, \quad (5)$$

as the mean value and the variance stay constant, i.e.  $\langle p(t) \rangle = \langle p(t + \Delta t) \rangle$  and  $\langle p(t)^2 \rangle = \langle p(t + \Delta t)^2 \rangle$ .

Applied to financial markets it can be observed that the autocorrelation function of price changes exhibits a significant negative value at time lag one tick, whereas it vanishes for time lags  $\Delta t > 1$ . Furthermore, the autocorrelation of absolute price changes or squared price changes, which is related to the volatility of the price process, decays slowly [15]. In order to implement (5) on a GPU architecture, similar steps as in section 3 are necessary. The calculation of the time lag-dependent part  $\langle p(t) \cdot p(t + \Delta t) \rangle$  is analogous to the determination of the Hurst exponent on the GPU. The input time series, which is transferred to the GPU's main memory, does not contain prices but price changes. However, in addition one needs the results for  $\langle p(t) \rangle$

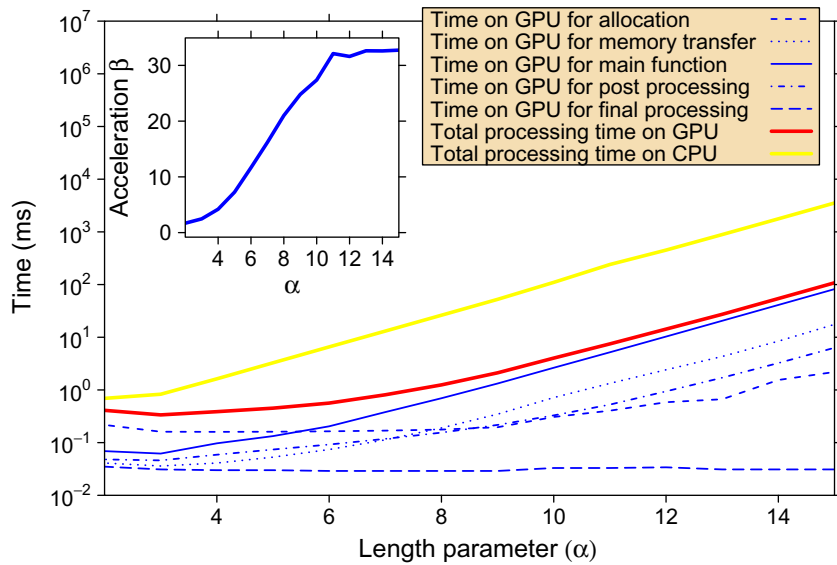


**Figure 7.** Hurst exponent  $H(\Delta t)$  in dependence of time lag  $\Delta t$  calculated on CPU and GPU. Additionally, the theoretical Hurst exponent of a random walk process ( $H = 0.5$ ) is included for comparison. One can clearly see the well-known anti-persistent behavior of the FGBL time series on short timescales ( $\Delta t < 2^4$  time ticks). Furthermore, evidence is given that the process reaches a slightly super-diffusive region ( $H \approx 0.525$ ) on medium timescales ( $2^4$  time ticks  $< \Delta t < 2^7$  time ticks). On long timescales, an asymptotic random walk behavior can be found. In order to quantify deviations from calculations on a CPU, the relative error  $\epsilon$  (see main text) is presented for each time lag  $\Delta t$  in the inset. It is typically smaller than  $10^{-3}$ .

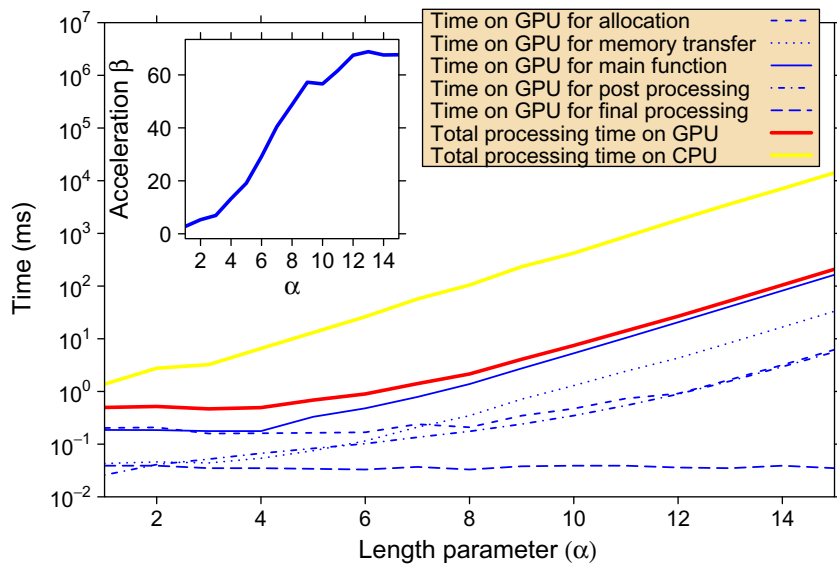
and  $\langle p(t)^2 \rangle$ . For this purpose, an additional array of length  $T$  is allocated, in which a GPU kernel function stores the squared values of the time series. Then, time series and squared time series are reduced with the same binary tree reduction process as in section 3. However, as this procedure produces arrays of length  $\Delta t_{\max}$ , one has to summarize these values in order to obtain  $\langle p(t) \rangle$  and  $\langle p(t)^2 \rangle$ .

The processing times for determining the autocorrelation function for  $\Delta t_{\max} = 256$  on CPU and 8800 GT can be found in figure 8. Here, we find that allocation and memory transfer dominate the total processing time on the GPU for small values of  $\alpha$  and thus, only a fraction of the maximum acceleration factor  $\beta \approx 33$ , which is shown as an inset, can be reached. Using the consumer graphics card GTX 280, we obtain a maximum speed-up factor of roughly 55 for  $\Delta t_{\max} = 256$  and 68 for  $\Delta t_{\max} = 512$  as shown in figure 9. In figure 10, the autocorrelation function of the FGBL time series is shown. At time lag one, the time series exhibits a large negative autocorrelation,  $\rho(\Delta t = 1) = -0.43$ . In order to quantify deviations between GPU and CPU calculations, the relative error  $\epsilon$  is presented in the inset of figure 10. Note that small absolute errors can cause relative errors up to three per cent because the values  $\rho(\Delta t > 1)$  are close to zero.

For some applications, it is interesting to study larger maximum time lags of the autocorrelation function. Based on our GPU implementation one has to modify the program code in the following way. So far, each thread was responsible for a specific time lag  $\Delta t$ .

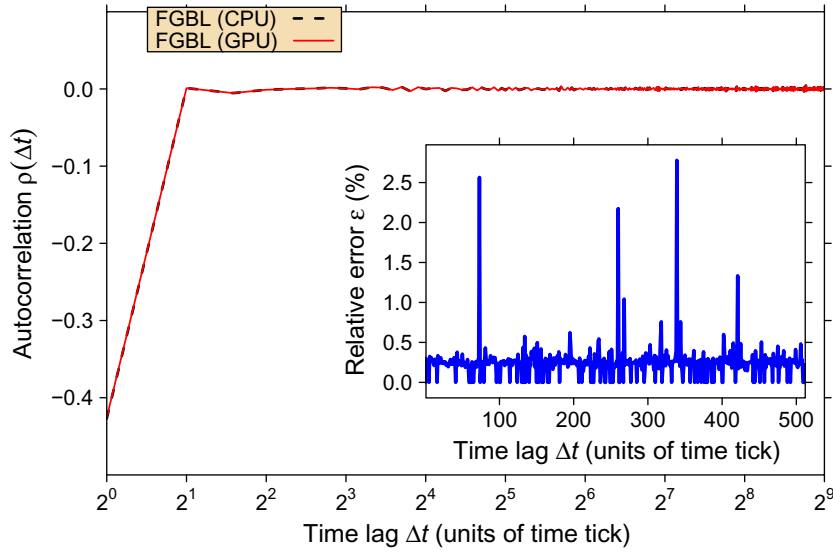


**Figure 8.** Processing times for the calculation of the equilibrium autocorrelation function  $\rho(\Delta t)$  on the GPU and CPU for  $\Delta t_{\max} = 256$ . The graphics card 8800 GT is used as the GPU device. The total processing time on the GPU is broken into allocation time, time for memory transfer, time for main processing, time for post-processing and time for final processing. The acceleration factor  $\beta$  is shown as inset. A maximum acceleration factor of roughly 33 can be obtained.



**Figure 9.** Processing times for the calculation of the equilibrium autocorrelation function  $\rho(\Delta t)$  on the GPU and CPU for  $\Delta t_{\max} = 512$ . The GTX 280 is used as the GPU device. A maximum acceleration factor of roughly 68 can be obtained.

In a modified ansatz, each thread is responsible for more than one time lag in order to realize a maximum time lag, which is a multiple of the maximal 512 threads per block. This way, one obtains a maximum speed-up factor of, e.g. roughly 84 for  $\Delta t_{\max} = 1024$  using the GTX 280.



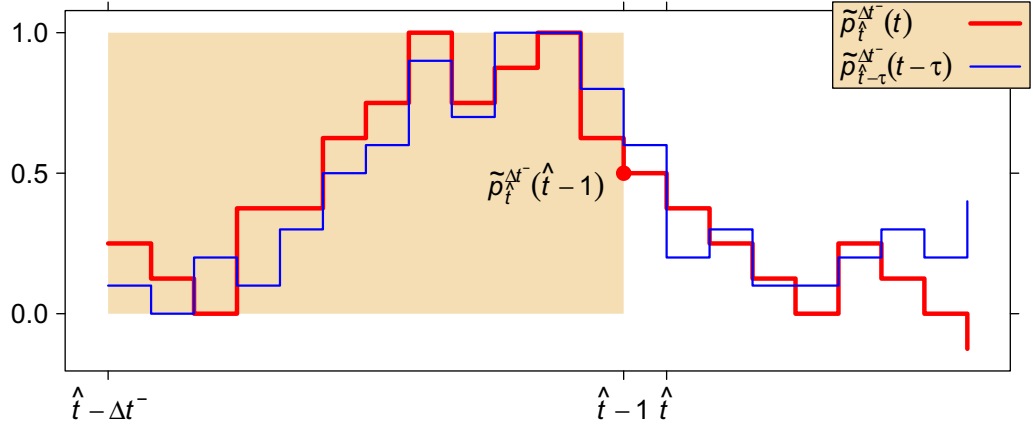
**Figure 10.** Equilibrium autocorrelation function  $\rho(\Delta t)$  in dependence of time lag  $\Delta t$  calculated on the CPU and GPU. One can clearly see the well-known negative autocorrelation of the FGBL time series at time lag one. In order to quantify deviations from calculations on a CPU the relative error  $\epsilon$  is presented for each time lag  $\Delta t$  in the inset. The relative error is always smaller than  $3 \times 10^{-2}$ .

## 5. Fluctuation pattern conformity

As a third method of time series analysis, the recently introduced fluctuation pattern conformity (PC) determination [50] was migrated to a GPU architecture. The PC quantifies pattern-based complex short-time correlations of a time series. In context of financial market time series, the existence of complex correlations implies that reactions of market participants to a given time series pattern are related to comparable patterns in the past. On medium and long timescales, one can state that no significant complex correlations can be measured because the price process exhibits random walk statistics. However, if one investigates the trading process on a tick-by-tick basis, evidence is given for recurring events. In the course of these considerations, a general pattern conformity observable is defined in [50], which is not limited to the application to financial market time series. In general, the aim is to compare a current pattern of time interval length  $\Delta t^-$  with all possible previous patterns of the time series  $p(t)$ . The current observation time shall be denoted by  $\hat{t}$ . Then, the current pattern's time interval measured in time ticks is given by  $[\hat{t} - \Delta t^-; \hat{t})$ . The evolution after this current pattern interval—the distance to  $\hat{t}$  is expressed by  $\Delta t^+$  (see below)—is compared with the prediction derived from all historical patterns. However, as the standard deviation of the price process is not constant in time, all comparison patterns have to be normalized with respect to the current pattern. For this purpose, the true range is used—the difference between high and low within each interval. Let  $p_h(\hat{t}, \Delta t^-)$  be the maximum value of a pattern of length  $\Delta t^-$  at time  $\hat{t}$  and analogously  $p_l(\hat{t}, \Delta t^-)$  be the minimum value. Thus, we can create a modified time series, which is true range adapted in the appropriate time interval, through

$$\tilde{p}_i^{\Delta t^-}(t) = \frac{p(t) - p_l(\hat{t}, \Delta t^-)}{p_h(\hat{t}, \Delta t^-) - p_l(\hat{t}, \Delta t^-)} \quad (6)$$

with  $\tilde{p}_i^{\Delta t^-}(t) \in [0; 1] \forall t \in [\hat{t} - \Delta t^-; \hat{t})$ , as illustrated in figure 11.



**Figure 11.** Schematic visualization of the pattern conformity estimation mechanism. The current pattern  $\tilde{p}_i^{\Delta t^-}(t)$  and the comparison pattern  $\tilde{p}_{i-\tau}^{\Delta t^-}(t-\tau)$  have the maximum value 1 and the minimum value 0 in  $[\hat{t} - \Delta t^-; \hat{t})$ , as shown by the filled rectangle. For the pattern conformity calculation, we need to analyze for each time difference  $\Delta t^+$  whether the current pattern value and the comparison pattern value at  $\hat{t} + \Delta t^+$  is above or below the last value of the current pattern  $\tilde{p}_i^{\Delta t^-}(\hat{t} - 1)$ . If both are above or below this last value, then +1 is added to the non-normalized pattern conformity  $\xi_\chi(\Delta t^+, \Delta t^-)$ . If one is above and the other below, then -1 is added.

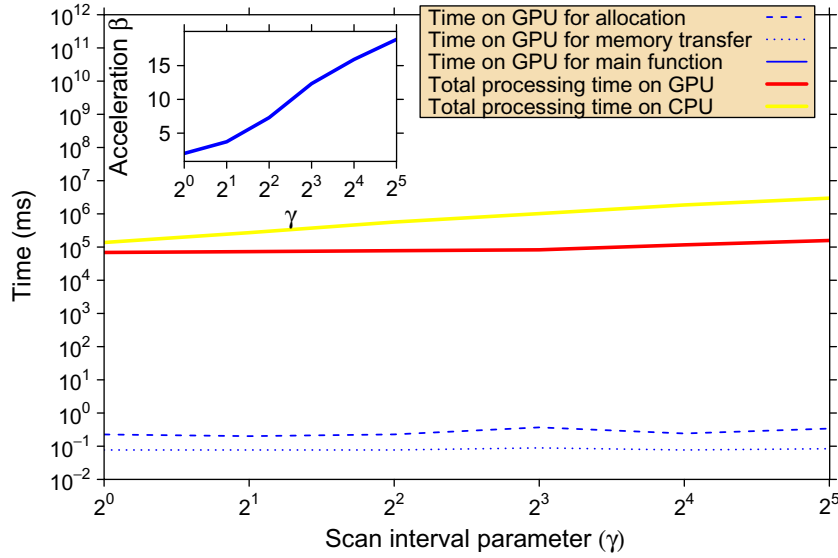
In order to assess the match of a pattern with a comparison pattern, the fit quality  $Q_i^{\Delta t^-}(\tau)$  between the current pattern sequence  $\tilde{p}_i^{\Delta t^-}(t)$  and a comparison pattern sequence  $\tilde{p}_{i-\tau}^{\Delta t^-}(t-\tau)$  for  $t \in [\hat{t} - \Delta t^-; \hat{t})$  has to be determined by the summation of the squared variations through

$$Q_i^{\Delta t^-}(\tau) = \sum_{\theta=1}^{\Delta t^-} \frac{\left( \tilde{p}_i^{\Delta t^-}(\hat{t} - \theta) - \tilde{p}_{i-\tau}^{\Delta t^-}(\hat{t} - \tau - \theta) \right)^2}{\Delta t^-}. \quad (7)$$

Note that  $Q_i^{\Delta t^-}(\tau)$  takes values in the interval  $[0, 1]$  as a result of the true range adaption. With these elements, one can define a pre-stage of the PC, which is not yet normalized, as motivated in figure 11, by

$$\xi_\chi(\Delta t^+, \Delta t^-) = \sum_{i=\Delta t^-}^{T-\Delta t^+} \sum_{\tau=\tau^*}^{\hat{t}} \frac{\text{sgn}(\omega_i^{\Delta t^-}(\tau, \Delta t^+))}{\exp(\chi Q_i^{\Delta t^-}(\tau))} \quad (8)$$

with  $\tau^* = \hat{t} - \hat{\tau}$  if  $\hat{t} - \hat{\tau} - \Delta t^- \geq 0$  and  $\tau^* = \Delta t^-$  else. In general, we limit the evaluation for each pattern to maximal  $\hat{\tau}$  historical patterns. Furthermore, for the sign function, we use the standard definition  $\text{sgn}(x) = 1$  for  $x > 0$ ,  $\text{sgn}(x) = 0$  for  $x = 0$ , and  $\text{sgn}(x) = -1$  for  $x < 0$ . In (8), the parameter  $\chi$  weighs pattern terms according to their qualities  $Q_i^{\Delta t^-}(\tau)$ . The larger  $\chi$ , the stricter the pattern weighting in order to use only pattern sequences with good agreement to the current pattern sequence. The expression  $\omega_i^{\Delta t^-}(\tau, \Delta t^+)$  in (8), which takes into account the value



**Figure 12.** Processing times for the calculation of the pattern conformity on GPU and CPU for  $\Delta t_{\max}^- = \Delta t_{\max}^+ = 20$ . The GTX 280 is used as GPU device. The total processing time on GPU is broken into allocation time, time for memory transfer, and time for main processing. The acceleration factor  $\beta$  is shown as inset. A maximum acceleration factor of roughly 19 can be obtained.

of current and comparison pattern sequences after  $\hat{t}$  for a proposed  $\Delta t^+$  relative to  $\tilde{p}_i^{\Delta t^-}(\hat{t} - 1)$ , is given by

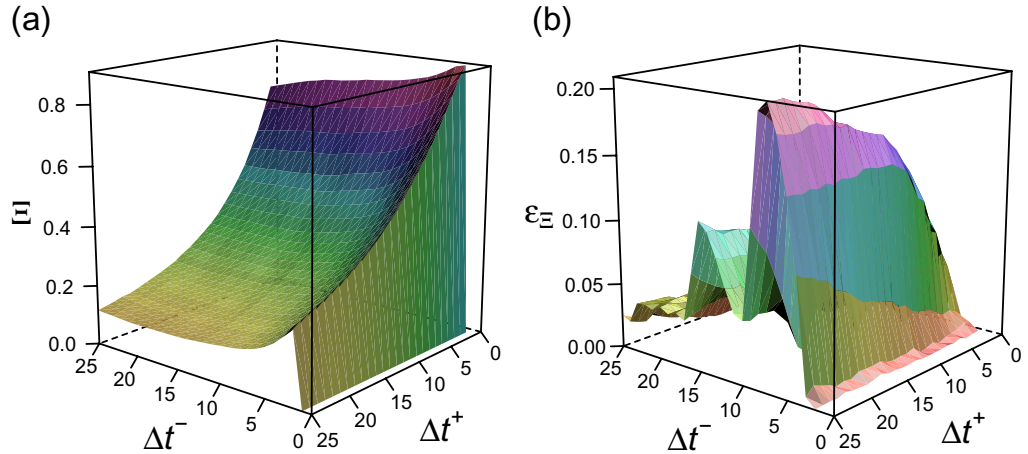
$$\omega_i^{\Delta t^-}(\tau, \Delta t^+) = \left( \tilde{p}_i^{\Delta t^-}(\hat{t} - 1 + \Delta t^+) - \tilde{p}_i^{\Delta t^-}(\hat{t} - 1) \right) \left( \tilde{p}_{i-\tau}^{\Delta t^-}(\hat{t} - \tau - 1 + \Delta t^+) - \tilde{p}_i^{\Delta t^-}(\hat{t} - 1) \right). \quad (9)$$

By normalizing (8) through its altered version, in which the sign function is replaced by its absolute value, the pattern conformity can be written as

$$\Xi_{\chi}(\Delta t^+, \Delta t^-) = \frac{\xi_{\chi}(\Delta t^+, \Delta t^-)}{\sum_{\hat{i}=\Delta t^-}^{T-\Delta t^+} \sum_{\tau=\tau^*}^{\hat{i}} \frac{|\text{sgn}(\omega_i^{\Delta t^-}(\tau, \Delta t^+))|}{\exp(\chi Q_i^{\Delta t^-}(\tau))}}. \quad (10)$$

We repeat that the pattern conformity is the most accurate measure to characterize the short-term correlations of a general time series. It is essentially given by the comparison of subsequences of the time series. Subsequences of various lengths are compared with historical sequences in order to extract similar reactions on similar patterns.

In order to realize a GPU implementation of the pattern conformity provided in (10), one has to allocate memory as for the Hurst exponent and for the autocorrelation function determination in sections 3 and 4, respectively. The allocation is needed for the array containing the time series, which has to be transferred to the global memory of the GPU, and for further processing arrays. The main processing GPU function is invoked with a proposed  $\Delta t^-$  and a given  $\hat{t}$ . In the kernel function, shared memory arrays for comparison and current pattern sequences are allocated and loaded from global memory of the GPU. In the main calculation, each thread handles one specific comparison pattern, i.e. each thread is responsible for one



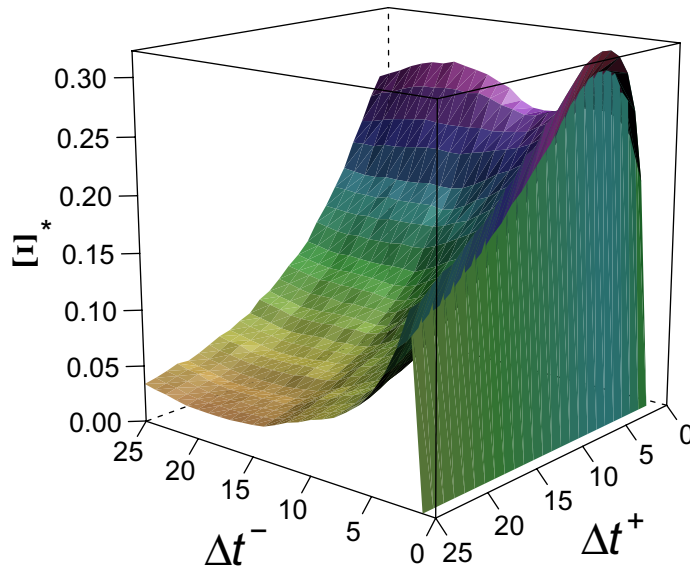
**Figure 13.** (a) Pattern conformity  $\Xi_{\chi=100}^{\text{GPU}}(\Delta t^-, \Delta t^+)$  of FGBL time series with  $\hat{\tau} = 16384$  calculated on the consumer graphics card GTX 280. (b) Relative error  $\epsilon_{\Xi}$  (in percent) between calculation on the GPU and CPU ( $\Xi_{\chi=100}^{\text{CPU}}(\Delta t^-, \Delta t^+)$ , with the same parameter settings). The processing time on the GPU was 5.8 h; the results on the CPU were obtained after 137.2 h, which corresponds to roughly 5.7 days. Thus, for these parameters an acceleration factor of roughly 24 is obtained.

value of  $\tau$  and so,  $\hat{\tau} = \gamma \times \sigma$  is applied with  $\gamma$  denoting the scan interval parameter and  $\sigma$  denoting the number of threads per block. Thus,  $\gamma$  corresponds to the number of blocks. The partial results of  $\xi_{\chi}(\Delta t^+, \Delta t^-)$  are stored in a global memory based array of dimension  $\hat{\tau} \times \Delta t^+$ . These partial results have to be reduced in a further processing step, which uses the same binary tree structure as applied in section 3 for the Hurst exponent determination.

The pattern conformity for a random walk time series, which exhibits no correlations by construction, is 0. The pattern conformity for a perfectly correlated time series is 1 [50]. A maximum speed-up factor of roughly 10 can be obtained for the calculation of the pattern conformity on the GPU and CPU for  $\Delta t_{\text{max}}^- = \Delta t_{\text{max}}^+ = 20$ ,  $T = 25000$ ,  $\chi = 100$  and  $\sigma = 256$  using the 8800 GT. In figure 12, corresponding results for using the GTX 280 are shown in dependence of the scan interval parameter  $\gamma$ . Here, a maximum acceleration factor of roughly 19 can be realized.

With this method, which is able to detect complex correlations of a time series, it is also possible to search for pattern conformity based complex correlations in financial market data, as shown in figure 13 for the FGBL time series. In figure 13(a), the results for the pattern conformity  $\Xi_{\chi=100}^{\text{GPU}}(\Delta t^-, \Delta t^+)$  are presented with  $\hat{\tau} = 16384$  calculated on the GTX 280. One can clearly see that for small values of  $\Delta t^-$  and  $\Delta t^+$  large values of  $\Xi_{\chi=100}^{\text{GPU}}$  are obtained with a maximum value of roughly 0.8. For the results shown in figure 13(b), the calculation of the pattern conformity is executed on the CPU, and in figure 13(c), the relative absolute error

$$\epsilon_{\Xi} = 10^2 \times \left| \frac{\Xi_{\chi=100}^{\text{GPU}} - \Xi_{\chi=100}^{\text{CPU}}}{\Xi_{\chi=100}^{\text{CPU}}} \right| \quad (11)$$



**Figure 14.** (a) FGBL pattern conformity corrected by the ACRW with  $\phi = 0.044$  and with  $1.5 \times 10^6$  time steps. Thus,  $\Xi^* = \Xi_{\chi=100}^{\text{FGBL}} - \Xi_{\chi=100}^{\text{ACRW}}$  is shown (see the text).

is shown, which is smaller than two-tenths of a per cent. This small error arises because the GPU device summarizes only a large number of the weighted values  $+1$  and  $-1$ . Thus, the limitation to single precision has no significant negative effect for the result.

This raw pattern conformity profile is dominated by trivial pattern correlation parts caused by the jumps of the price process between best bid and best ask price—the best bid price is given by the highest limit order price of all buy orders in an order book and analogously the best ask price is given by the lowest limit order price of all sell orders in an order book. As performed in [50], there are possibilities for reducing these trivial pattern conformity parts. For example, it is possible to add such jumps around the spread synthetically to a random walk. Let  $p_\phi^*$  be the time series of the synthetically anti-correlated random walk created (ACRW) in a Monte Carlo simulation through  $p_\phi^* = a_\phi(t) + b(t)$ , which was used in sections 3–5 as synthetic time series. With probability  $\phi \in [0; 0.5]$  the expression  $a_\phi(t+1) - a_\phi(t) = +1$  will be applied and with probability  $\phi$  a decrement  $a_\phi(t+1) - a_\phi(t) = -1$  will occur. With probability  $1-2\phi$  the expression  $a_\phi(t+1) = a_\phi(t)$  is used. The stochastic variable  $b(t)$  models the bid-ask spread and can take the value 0 or 1 in each time step, each with probability 0.5. Thus, by changing  $\phi$ , the characteristic timescale of process  $a_\phi$  in comparison to process  $b$  can be modified.

Parts of the pattern-based correlations in figure 13 stem from this trivial negative autocorrelation for  $\Delta t = 1$ . In order to try to correct for this, in figure 14 (an animated visualization can be found in the multimedia enhancements of this publication), the pattern conformity of the ACRW with  $\phi = 0.044$ , which reproduces the anti-correlation of the FGBL time series at time lag  $\Delta t = 1$ , is subtracted from the data of figure 13(a). Obviously, the autocorrelation for the time lag  $\Delta t = 1$ , which is understood from the order book structure is not the sole reason for the pattern formation conformity, which is shown in figure 13(a). Thus, evidence is obtained that financial market time series show pattern correlation on very short timescales beyond the simple anti-persistence which is due to the gap between bid and ask prices.

## 6. Conclusion and outlook

In this paper, we applied the compute unified device architecture—a programming approach for issuing and managing computations on a GPU as a data-parallel computing device—to methods of fluctuation analysis. Firstly, the Hurst exponent calculation was presented performed on a GPU. These results of the scaling behavior of a stochastic process can be obtained up to 80 times faster than on a current CPU core and the relative absolute error of the results obtained from the CPU and GPU is smaller than  $10^{-3}$ . The calculation of the equilibrium autocorrelation function was also migrated to a GPU device successfully and applied to a financial market time series. In this case, acceleration factors up to roughly 84 were realized. In a further part, the pattern formation conformity algorithm, which quantifies pattern-based complex short-time correlations of a time series, was determined on a GPU. For this application the GPU was up to 24 times faster than the CPU, and the values provided by the GPU and CPU differ only in a relative error of maximal two-tenths of a per cent. Furthermore, we could verify, that the current GPU generation is roughly two times faster than the previous one. The presented methods were applied to an FGBL time series of the Eurex, which exhibits an anti-persistent regime on short timescales. Evidence was found that a super-diffusive regime is reached on medium timescales. On long timescales, the FGBL time series complies to random walk statistics. Furthermore, the anti-correlation at time lag one—an empirical stylized fact of financial market time series—was verified. The pattern conformity which is used is the most accurate measure to characterize the short-term correlations of a general time series. It is essentially given by the comparison of subsequences of the time series. Subsequences of various lengths are compared with historical sequences in order to extract similar reactions on similar patterns. The pattern conformity of the FGBL contract exhibits large values up to 0.8. However, these values also include the trivial auto-correlation property at time lag one, which can be removed by the pattern conformity of a synthetic anti-correlated random walk. However, significant pattern based correlations are still exhibited after correction. Thus, evidence is obtained that financial market time series show pattern correlation on very short timescales beyond the simple anti-persistence, which is due to the gap between bid and ask prices. Further applications of the GPU-accelerated techniques in context of the Monte Carlo simulations and agent-based modeling of financial markets are certainly well worth pursuing. As already mentioned in the introduction, the main advantage of general-purpose computations on GPUs is that one does not need special-purpose computers. Although GPU computing opens a large variety of possibilities, the recent development of using graphic cards for scientific computing will perhaps also revive special-purpose computing as GPU implementations are not appropriate for each problem.

## Acknowledgments

This work was financially supported by the German Research Foundation (DFG) and benefited by the Forschungsfond of the Materialwissenschaftliches Forschungszentrum (MWFZ) of the Johannes Gutenberg University of Mainz. The present work is based on the private opinion of the authors and does not necessarily reflect the views of Artemis Capital Asset Management GmbH.

## Appendix. GPU source code

Source code of the GPU kernel functions for determining the time lag-dependent Hurst exponent  $H(\Delta t)$ . These functions are executed on the GPU device.

```

/****
 *
 * Device function main
 *
 */
__global__ void dev(float* in, float* out) {
    __shared__ float in_s[2*BLOCK_SIZE];
    __shared__ float out_s[BLOCK_SIZE];

    in_s[threadIdx.x]=in[blockIdx.x*BLOCK_SIZE+threadIdx.x];
    in_s[BLOCK_SIZE+threadIdx.x]=in[BLOCK_SIZE+blockIdx.x*BLOCK_SIZE+threadIdx.x];

    out_s[threadIdx.x]=0;
    __syncthreads();

    for(int t=0;t<BLOCK_SIZE;++t) {
        out_s[threadIdx.x]+=fabs(in_s[t+threadIdx.x]-in_s[t]);
    }

    __syncthreads();
    out[blockIdx.x*BLOCK_SIZE+threadIdx.x]=out_s[threadIdx.x];
}

/****
 *
 * Device function post processing
 *
 */
__global__ void dev_postprocessing(float* in, int offset) {
    __shared__ float in_s[2*BLOCK_SIZE];
    in_s[threadIdx.x]=in[2*blockIdx.x*offset+threadIdx.x];
    in_s[BLOCK_SIZE+threadIdx.x]=in[2*blockIdx.x*offset+offset+threadIdx.x];

    __syncthreads();

    in_s[threadIdx.x]=in_s[threadIdx.x]+in_s[BLOCK_SIZE+threadIdx.x];
    in[2*blockIdx.x*offset+threadIdx.x]=in_s[threadIdx.x];
}

/****
 *
 * Device function final processing
 *
 */
__global__ void dev_finalprocessing(float* in, float* out, int in_size) {
    __shared__ float in_s[BLOCK_SIZE];
    if(threadIdx.x>0) in_s[threadIdx.x]=log10(in[threadIdx.x]/(in_size-BLOCK_SIZE));

    __syncthreads();

    if(threadIdx.x>1) out[threadIdx.x]=(in_s[threadIdx.x]-in_s[threadIdx.x-1])/
        (log10((float)(threadIdx.x))-log10((float)(threadIdx.x-1)));
}

```

## References

- [1] Black F and Scholes M 1973 *J. Polit. Econ.* **81** 637
- [2] Cont R and Bouchaud J P 2000 *Macroecon. Dyn.* **4** 170
- [3] Gopikrishnan P, Plerou V, Amaral L A N, Meyer M and Stanley H E 1999 *Phys. Rev. E* **60** 5305
- [4] Mantegna R N and Stanley H E 2000 *An Introduction to Econophysics—Correlations and Complexity in Finance* (Cambridge: Cambridge University Press)
- [5] Bouchaud J P and Potters M 2000 *Theory of Financial Risks—From Statistical Physics to Risk Management* (Cambridge: Cambridge University Press)

- [6] Paul W and Baschnagel J 2000 *Stochastic Processes: From Physics to Finance* (Heidelberg: Springer)
- [7] Mandelbrot B 1963 *J. Bus.* **36** 394
- [8] Mandelbrot B 1964 *J. Bus.* **37** 393
- [9] Kiyono K, Struzik Z R and Yamamoto Y 2006 *Phys. Rev. Lett.* **96** 068701
- [10] Bouchaud J P, Matacz A and Potters M 2001 *Phys. Rev. Lett.* **87** 228701
- [11] Krawiecki A, Holyst J A and Helbing D 2002 *Phys. Rev. Lett.* **89** 158701
- [12] Daniels M G, Farmer J D, Gillemot L, Iori G and Smith E 2003 *Phys. Rev. Lett.* **90** 108102
- [13] Smith E, Farmer J D, Gillemot L and Krishnamurthy S 2003 *Quant. Finance* **3** 481
- [14] Preis T, Golke S, Paul W and Schneider J J 2006 *Europhys. Lett.* **75** 510
- [15] Preis T, Golke S, Paul W and Schneider J J 2007 *Phys. Rev. E* **76** 016108
- [16] Bak P, Paczuski M and Shubik M 1997 *Physica A* **246** 430
- [17] Challet D and Stinchcombe R 2003 *Quant. Finance* **3** 155
- [18] Maslov S 2000 *Physica A* **278** 571
- [19] Maslov S and Mills M 2001 *Physica A* **299** 234
- [20] Stauffer D and Penna T J P 1998 *Physica A* **256** 284
- [21] Stauffer D and Penna T J P 1999 *Physica A* **271** 496
- [22] Lux T and Marchesi M 1999 *Nature* **397** 498
- [23] Laloux L, Cizeau P, Bouchaud J P and Potters M 1999 *Phys. Rev. Lett.* **83** 1467
- [24] Plerou V, Gopikrishnan P, Rosenow B, Amaral L A N and Stanley H E 1999 *Phys. Rev. Lett.* **83** 1471
- [25] O'Hara M 1997 *Market Microstructure Theory* (Malden, MA: Blackwell)
- [26] Biais B, Hillion P and Spatt C 1995 *J. Finance* **50** 1655
- [27] Brunnermeier M and Pedersen L 2005 *J. Finance* **60** 1825
- [28] Bertsimas D and Lo A 1998 *J. Financ. Markets* **1** 1
- [29] Bank P and Baum D 2004 *Math. Finance* **14** 1
- [30] Landau D and Binder K 2005 *A Guide to Monte Carlo Simulations in Statistical Physics* (Cambridge: Cambridge University Press)
- [31] van Meel J A, Arnold A, Frenkel D, Zwart S P and Belleman R G 2008 *Mol. Simul.* **34** 259
- [32] Köstler H, Schmid R, Rude U and Scheit C 2008 *Comput. Vis. Sci.* **11** 115
- [33] Schneider J J and Kirkpatrick S 2006 *Stochastic Optimization* (Berlin: Springer)
- [34] Dagum L and Menon R 1998 *IEEE Comput. Sci. Eng.* **5** 46
- [35] Gabriel E *et al* 2004 Open MPI: goals, concept and design of a next generation MPI implementation *Proc. 11th European PVM/MPI Users' Group Meeting (Budapest, Hungary)* pp 97–104
- [36] Tomov S, McGuigan M, Bennett R, Smith G and Spiletic J 2005 *Comput. Graph.* **29** 71
- [37] Stone J E, Phillips J C, Freddolino P L, Hardy D J, Trabuco L G and Schulten K 2007 *J. Comput. Chem.* **28** 2618
- [38] Susukita R *et al* 2003 *Comput. Phys. Commun.* **155** 115
- [39] Zwart S F P, Bellemana R G and Geldof P M 2007 *New Astron.* **12** 641
- [40] Bellemana R G, Bédorf J and Zwart S F P 2007 *New Astron.* **13** 103
- [41] Li W, Wei X and Kaufman A 2003 *Vis. Comput.* **19** 444
- [42] Anderson J A, Lorenz C D and Travesset A 2008 *J. Comput. Phys.* **227** 5342
- [43] Yanga J, Wang Y and Chen Y 2007 *J. Comput. Phys.* **221** 799
- [44] Preis T, Virnau P, Paul W and Schneider J J 2009 *J. Comput. Phys.* **228** 4468
- [45] Rost R J 2006 *OpenGL Shading Language* 2nd edn (Reading, MA: Addison-Wesley)
- [46] Fernando R and Kilgard M J 2003 *The Cg Tutorial: The Definitive Guide to Programmable Real-time Graphics* (Reading, MA: Addison-Wesley)
- [47] NVIDIA *CUDA Compute Unified Device Architecture, Programming Guide* 2008 version 2.0 NVIDIA Corporation (<http://www.nvidia.com>)
- [48] ATI *CTM Guide, Technical Reference Manual* 2006 version 1.01 Advanced Micro Devices, Inc. (<http://www.amd.com>)

- [49] *NVIDIA GeForce GTX 280 Specifications* 2008 NVIDIA Corporation (<http://www.nvidia.com>)
- [50] Preis T, Paul W and Schneider J J 2008 *Europhys. Lett.* **82** 68005
- [51] Mandelbrot B and Hudson R 2004 *The (mis)Behavior of Markets. A Fractal View of Risk, Ruin and Reward* (New York: Basic Books)
- [52] Hurst H E 1951 *Trans. Am. Soc. Civil Engin.* **116** 770
- [53] Darbellay G A and Wuertz D 2000 *Physica A* **287** 429
- [54] Ausloos M 2000 *Physica A* **285** 48
- [55] Carbone A, Castelli G and Stanley H E 2004 *Physica A* **344** 267
- [56] Gu G F and Zhou W X 2009 *Eur. Phys. J. B* **67** 585