ISSN 1392–124X (print), ISSN 2335–884X (online) INFORMATION TECHNOLOGY AND CONTROL, 2015, T. 44, Nr. 2

Parallel Computing for Mixed-Stable Modelling of Large Data Sets

Igoris Belovas

Vilnius University Institute of Mathematics and Informatics, Optimization Sector at Systems Analysis
Akademijos str., 4, LT-08663 Vilnius, Lithuania
e-mail: igoris.belovas@mii.vu.lt

Vadimas Starikovičius

Vilnius Gediminas Technical University, Laboratory of Parallel Computing
Sauletekio al. 11, LT-10223 Vilnius, Lithuania
e-mail: vadimas.starikovicius@vgtu.lt

crossref http://dx.doi.org/10.5755/j01.itc.44.2.6723

Abstract. In this paper, we develop efficient parallel algorithms for the statistical processing of large data sets. Namely, we parallelize the maximum likelihood method for the estimation of parameters of the mixed-stable model. This method is known to be very computationally demanding. Financial German DAX stock index data are used as empirical data in this work. Several hierarchical levels of parallelism were distinguished, analyzed and implemented using OpenMP and MPI library. Parallel performance tests were conducted on the IBM SP6 supercomputer. Obtained performance results show that implemented parallel algorithms are very efficient and scalable on distributed and shared memory systems. Speedups up to 800 times were obtained for 1024 parallel processes. Noticeably, our parallel application is able to efficiently utilize the Simultaneous multithreading (Intel Hyper-Threading) technology in modern processors. This research demonstrates that the application of modern parallel technologies allows a fast and accurate estimation of mixed-stable parameters even for large amounts of data and promotes a wider use of stable modelling for the statistical data processing.

Keywords: parallel computing and algorithms; simultaneous multithreading (SMT); large data sets; high-frequency data; mixed-stable model; financial modelling.

1. Introduction

Fast and efficient processing of huge amounts of diverse data is essential in our Information Age. The speed and accuracy of data processing is vital in modern economy and finance. Contemporary electronic commerce and trading tools provide economists and market analysts with large amounts of information. Reliable and well-timed business decision-making based on this information is virtually impossible without the application of parallel computing [8, 11].

Among the established and popular instruments in the statistical modelling are stable distributions. They are applied in such diverse areas as astronomy, physics, electronics, sociology, economics [19]. Since the second half of the 20th century a particularly important role they play in financial mathematics [17]. In our previous works we have employed stable distributions for the modelling of stock and foreign exchange markets [1, 9]. It should be noted that the practical application of stable distributions is hampered by the absence of analytical representation of the general probability density function. This makes the application of stable models computationally demanding.

The most accurate method for the evaluation of parameters of a stable model is the maximum likelihood method. At the same time it is the most time-consuming [9]. Therefore it is often rejected [10]. However, in this work we will demonstrate that the application of modern parallel technologies makes this approach both precise and practical. In this research, we develop and expand our parallel computing algorithms [2] to the mixed-stable modelling of high-frequency data. To our best knowledge, such an approach to statistical processing of big data was not used before. Albeit parallel technologies are intensively used with other statistical models for big data processing [6, 7, 20]. Financial German DAX

stock index data are used as empirical data in this work. It should by emphasized that the presented approach can be applied for the statistical processing of empirical data of diverse origins.

The paper is organized as follows. The first part is the introduction. In Section 2, we present our modelling methodology briefly introducing stable distributions, maximal likelihood method and mixed-stable model. Section 3 describes the parallelization of our algorithms and demonstrates its efficiency and scalability. Conclusions are drawn is Section 4.

2. Stable and mixed-stable modelling

2.1. α-stable modelling

The $\alpha-$stable distribution depends on four parameters and is usually described by its characteristic function $\varphi(t)$:

$$\log \varphi(t) = \begin{cases} -\sigma^{\alpha}|t|^{\alpha}\{1-i\beta\frac{t}{|t|}\tan\frac{\pi\alpha}{2}\}+i\mu t, & \alpha \neq 1 \\ -\sigma|t|\{1+i\beta\frac{2t}{\pi|t|}\log|t|\}+i\mu t, & \alpha = 1 \end{cases}$$

where $\alpha \in (0,2], \beta \in [-1,1]$, $\sigma > 0$, $\mu \in \mathsf{R}$. Here α is the stability index, β is the skewness, μ is the location parameter and σ is the scale parameter. In financial modelling it is generally assumed [9] that $\alpha > 1$. The overview of properties of stable distributions can be found in [1, 19]. Noticeably, the probability density function of stable laws has no analytical representation, i.e. cannot be expressed in elementary functions, except for a few cases: Levy, Cauchy, and Gaussian distributions.

The most accurate method for the evaluation of parameters of a stable model is the maximum likelihood method [9, 15]. The vector of stable parameters $\Theta = (\alpha, \beta, \mu, \sigma)$ can be estimated from the set of returns $\{X_k\}$ by maximizing the log-likelihood function

$$L(\Theta) = \sum_{k=1}^{n} \ln\, p(X_k, \Theta), \qquad (2)$$

where

$$p(x,\Theta) = \frac{1}{\pi\sigma}\int_0^{\infty} e^{-t^{\alpha}} \cos h_{\alpha,\beta}\left(\frac{x-\mu_0}{\sigma}, t\right)dt, \qquad (3)$$

$$h_{\alpha,\beta}(x,t) = \begin{cases} xt + \beta\tan\frac{\pi\alpha}{2}(t - t^{\alpha}), & \alpha \neq 1, \\ xt - \beta t\frac{2}{\pi}\ln t, & \alpha = 1, \end{cases}$$

and

$$\mu_0 = \begin{cases} \mu + \beta\sigma\tan\frac{\pi\alpha}{2}, & \alpha \neq 1, \\ \mu + \beta\sigma\frac{2}{\pi}\ln\sigma, & \alpha = 1. \end{cases}$$

A precise and fast calculation of values $p(X_k, \Theta)$ of probability density function is a nontrivial task, see [9]. Obviously, it is a crucial part of the whole computational algorithm. In this work, the improper integral (3) is approximated by the definite one using the smart-Δ approach and adaptively calculated using Gauss-Kronrod 21-point integration rule [12].

2.2. Mixed-stable modelling

In this research as empirical data we have used the high-frequency series of returns of 29 stocks from the DAX (Deutscher AktienindeX) German stock index. These series are representing intra-daily data of the whole year: from January 1, 2007 to December 27, 2007; i.e. 251 days. We have aggregated the raw inhomogeneous intra-daily data into the equally-spaced homogeneous intra-daily time series. The aggregation is done with the previous-tick interpolation [21].

Having processed yearly high-frequency series of returns for all 29 stocks at different time steps, we have obtained that almost all data series are asymmetric and the empirical kurtosis shows that density functions of those series are more peaked than Gaussian density functions. This implies the utilization of stable distributions.

However, it should be pointed out, that these empirical data often exhibit the stagnation effect, i.e. series contain numerous zero returns. For example, at 10-second time step yearly series of returns contain from 43% to 82% of zeros and the length of series varies from 135001 to 436143 (with zeros removed). To deal with this zero effect, we apply to our data the mixed-stable model intoduced by I. Belovas et al. [9].

The probability density function of a mixed-stable random variable is

$$f(x,\Theta) = (1-r)\,p(x,\Theta) + r\delta(x) \qquad (4)$$

where $p(x,\Theta)$ is the probability density function (3) of a α-stable distribution (1) and $\delta(x)$ is the Dirac delta function. The coefficient $r \in (0,1)$ is the index of stagnation. The empirical cumulative distribution functions of data series with the stagnation effect exhibits jump at $x = 0$. Model (4) enables us to accommodate for these jumps.

The likelihood function of the mixed-stable model (4) is given by

$$l(r,\Theta) \sim (1-r)^{n-k} r^{k} \prod_{j=1}^{n-k} p(x_j, \Theta)$$

where $\{x_1, x_2, \ldots, x_{n-k}\}$ is a non-zero returns set, obtained by excluding k zero returns from the initial

returns set $\{X_1, X_2,\ldots, X_n\}$. By optimizing the first factor $(1 - r)^{n-k}r^k$, we obtain $r_{max} = k / n$. Optimization of the product is equivalent to the optimization of the likelihood function of the stable distribution of the non-zero returns set $\{x_1, x_2,\ldots, x_{n-k}\}$. Hence, the optimal vector Θ_{max} is estimated with non-zero returns via stable log-likelihood function (2).

To optimize the log-likelihood function, we use the Nelder-Mead method [14]. Although this method is not the fastest one, but it is robust and does not require any derivative (gradient, Hessian) calculation.

3. Parallelization

Numerical computations by the computationally demanding maximum likelihood method combined with the nontrivial numerical calculation of a stable probability density function are very time-consuming, especially when we need to handle long sets of high-frequency data. Nowadays the common approach to many time-consuming computational problems is the application of parallel computing technologies. In our previous work [2], we have already shown that the stable modelling technique is very suitable for parallelization. Several hierarchical levels of parallelism can be distinguished and defined: processing of multiple data sets (MS); algorithmic steps of optimization method (OPT); calculation of values of maximum likelihood (ML) target function (2); calculation of integral (3) of probability density function (PDF).

The coarse-grained parallelization at the MS level is easily implemented by a uniform distribution of data sets among the available parallel processes for independent ML optimization tasks. The MPI [13] implementation of this parallel algorithm showed almost perfect efficiency on the commodity PC cluster [2]. However, these results were obtained for artificially generated data sets of the same size. Moreover, all data sets were generated with the single set of stable parameters and different seeds. Obviously, working with the real data we need to deal with data sets of different sizes (see Section 2) and different computational complexities (number of iterations) of ML optimization. The resulting load balancing problems will definitely reduce the efficiency of parallelization and can be only partially overcome using heuristic scheduling algorithms for optimal distribution of data sets between parallel processes.

Another restraint of parallelization at the MS level is its low degree of concurrency. The maximal number of parallel processes, which can be utilized, is equal to the number of data sets to be analyzed. In this study, it would be 29 parallel processes for the data sets from all 29 stocks and a fixed time step. It is too little for the current parallel computing systems, although the introduction of stochastic or deterministic global optimization can relax this limitation. And finally, parallelization at the MS level does not reduce the time of processing of a single data set, which is vital for the time-critical applications in the financial analysis.

Therefore, in this research, we have chosen a more fine-grained parallelization at the ML target function (2) calculation level. In our previous work [2], we have shown that OpenMP [16] implementation of the parallel sum calculation is very efficient on the SMP (Symmetric multiprocessing) node with two Intel Pentium III processors. Since then we have a fully established era of multicore processors, that are currently used not only in the computing nodes of supercomputers, but also in the personal computers. The number of physical cores is constantly increasing. However, multicore processors are known to have scalability problems due to the memory bandwidth bottleneck [4]. For many memory intensive applications the efficiency and scalability of such parallelization are quite low [18].

Therefore, in this work, we have first studied the scalability of parallelization at the ML level using the OpenMP on the multicore architecture. Parallel performance tests were conducted on the IBM SP6 Power6 575 computing node with 32 cores and 128 GB of shared memory. The obtained results are shown in Table 3 and Figure 1. All parallel performance tests in this section were performed for two data sets obtained from one of the DAX stock's (Allianz SE) real-time data. A relatively small data set ($n = 7385$) was obtained using the previous-tick interpolation with time step $\Delta t = 1000$ and large data set ($n = 436143$) with time step $\Delta t = 10$. For all tests, the accuracy of ML optimization with the Nelder-Mead simplex method [14] was set to 10^{-6}. The integral (3) of stable probability density function was computed with the accuracy 10^{-11}.

The OpenMP programming standard [16] allows to specify a scheduling method for the assignment of summands (2) (i.e. loop iterations) to the parallel processes (threads). We have found that the choice of the scheduling method has a significant impact on the performance of our parallel application. It appears that the static schedule with the chunk of one summand is better for the small data set, when the additional overhead of dynamic assignment does not pay off. On the other hand, for the large data set, the dynamic assignment of chunks of 100 summands produces better results, because it is more evenly distributing the workload between the parallel processes. Note that the calculation of each summand in (2) can be of different computational complexity due to adaptive integration of PDF integral (3).

Further analyzing the Table 3, we see that our parallel algorithm scales very well up to 32 cores. Natural degradation of parallelization efficiency with the increasing number of parallel processes is even smaller for the large data set. Obviously, in our modelling algorithm, the ratio between the computations and data movement to and from the shared memory is much more favourable to the

Table 1. Performance and scalability of ML parallelization with the OpenMP on multicore IBM Power6 575 node. P - number of processes, T_p - processing time with p processes, S_p- speedup, E_p - efficiency

P	1	2	4	8	16	32	64
			Δt = 1000, set size n = 7385, static schedule (1)				
T_p	80.13	43.73	21.89	10.99	5.58	2.98	2.01
S_p	-	1.83	3.66	7.29	14.36	26.89	39.87
E_p	-	92%	92%	91%	90%	84%	62%
			Δt = 1000, set size n = 7385, dynamic schedule (100)				
T_p	80.13	43.74	22.31	11.72	6.24	3.96	3.28
S_p	-	1.83	3.59	6.84	12.84	20.24	24.43
E_p	-	92%	90%	86%	80%	63%	38%
			Δt = 10, set size n = 436143, static schedule (1)				
T_p	6598.01	3518.15	1784.11	893.04	453.04	231.34	118.23
S_p	-	1.88	3.70	7.39	14.56	28.52	55.81
E_p	-	94%	93%	92%	91%	89%	87%
			Δt = 10, set size n = 436143, dynamic schedule (100)				
T_p	6598.01	3560.43	1733.52	868.10	435.23	218.86	111.24
S_p	-	1.85	3.81	7.60	15.16	30.15	59.31
E_p	-	93%	95%	95%	95%	94%	93%

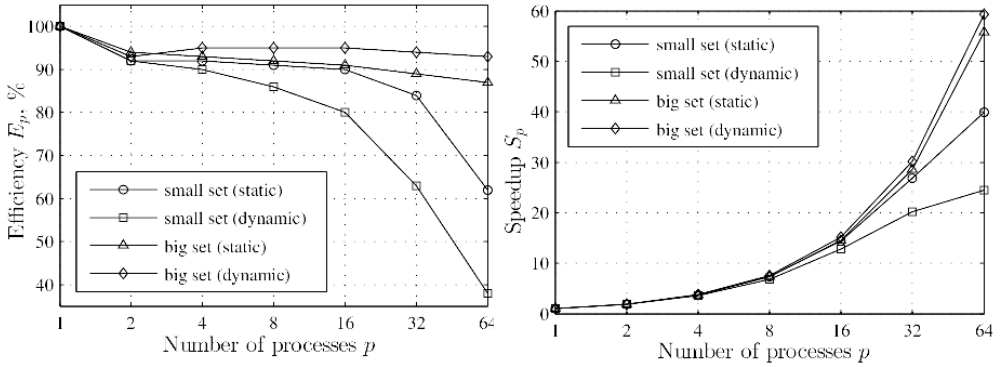

Figure 1. Speedup and efficiency of ML parallelization with the OpenMp on multicore IBM Power6 575 node

Table 2. Performance and scalability of ML parallelization with the MPI on multicore IBM Power6 575 node. P - number of processes, T_p - processing time with p processes, S_p- speedup, E_p - efficiency

P	1	2	4	8	16	32	64
			Δt = 1000, set size n = 7385				
T_p	80.13	40.51	20.31	10.29	5.30	2.80	1.50
S_p	-	1.98	3.95	7.79	15.12	28.64	53.42
E_p	-	99%	99%	97%	95%	90%	84%
			Δt = 10, set size n = 436143				
T_p	6589.13	3330.50	1657.39	826.23	418.94	210.81	120.38
S_p	-	1.98	3.98	7.99	15.75	31.30	54.81
E_p	-	99%	100%	100%	98%	98%	86%

Table 3. Performance and scalability of ML parallelization with the MPI on IBM SP6 using nodes with SMT (i.e. 64 processes per node). N d - number of nodes, P - number of processes, T_p - processing time with p processes, S_p- speedup, E_p - efficiency

N d	2	3	4	6	8	12	16
P	128	192	256	384	512	768	1024
			Δt = 1000, set size n = 7385				
T_p	0.791	0.563	0.421	0.300	0.248	0.186	0.163
S_p	101.30	142.42	190.33	267.11	323.11	431.79	492.98
E_p	79%	74%	74%	70%	63%	56%	48%
			Δt = 10, set size n = 436143				
T_p	60.85	40.68	28.78	19.66	15.08	10.75	8.26
S_p	108.43	162.20	229.26	335.58	437.53	613.94	798.71
E_p	85%	85%	90%	87%	86%	80%	78%

Table 4. Performance and scalability of ML parallelization with the MPI on IBM SP6 using nodes without SMT (i.e. 32 processes per node). N d - number of nodes, P - number of processes, T_p - processing time with p processes, S_p- speedup, E_p - efficiency

N d	2	4	6	8	10	12	16
P	64	128	192	256	320	384	512
			Δt = 1000, set size n = 7385				
T_p	1.388	0.728	0.516	0.382	0.332	0.273	0.223
S_p	57.73	110.07	155.16	209.76	241.46	293.40	359.48
E_p	90%	86%	81%	82%	76%	76%	70%
			Δt = 10, set size n = 436143				
T_p	107.05	55.42	36.50	27.86	22.29	18.80	14.03
S_p	61.64	119.06	180.79	236.83	295.95	350.96	470.13
E_p	96%	93%	94%	93%	93%	91%	92%

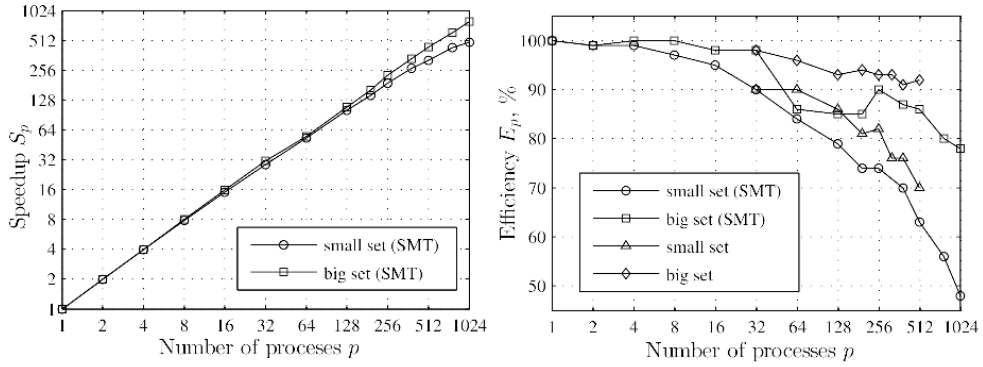

Figure 2. Speedup and efficiency of ML parallelization with the MPI on IBM SP6

current multicore architectures than in modelling algorithms based on the numerical solution of partial differential equations [18]. Finally, we have used the IBM processor's hardware support for the Simultaneous Multi-Threading (SMT), which can double the number of processes per core, i.e. up to 64 processes per node. The obtained performance gains are as significant as one might expect from the doubling of the number of physical cores. Hence, the SMT or Intel Hyper-Threading mode in the current processors cannot be overseen and should be definitely used for running our parallel application.

Next, to test further the scalability of chosen approach, we have implemented parallelization at the

ML level using the MPI library [13]. First, we have tested the MPI implementation on the same shared memory multicore node (IBM Power6 575) to compare it with the OpenMP implementation. The obtained performance results are shown in Table 2. Quite surprisingly, we see that the MPI implementation is slightly outperforming the OpenMP implementation on the shared memory architecture for both small and large data sets, despite the fact that the MPI parallelization is using static distribution of summands (2) between processes. Obviously, for our application, the IBM MPI library deals better with the shared memory contention than the IBM OpenMP compiler.

The main advantage of the MPI implementation is a feasibility to use the parallel computing systems with the distributed memory and hence with the bigger number of parallel processors. Next, we have tested the scalability of MPI parallelization at ML level on the IBM SP6 supercomputer in CINECA, Italy. The obtained results are presented in Tables 3, 4, and Figure 5. In Table 3, we present the results obtained using the SMT mode on multicore nodes, i.e. with 64 parallel processes per node. As we see, we get a speedup even for the large number of processes and small data set. As the number of summands (2) per process is reducing to less than 10, the efficiency is degrading due to the relative increase in parallelization overhead and imbalance in the workload. For the large data set, the efficiency is very good.

In Table 4, we present the results obtained without the SMT mode, i.e. with 32 parallel processes per node using only the physical cores. Naturally, the obtained speedup S_p and efficiency E_p numbers are better than corresponding numbers in Table 3. Without the effect of SMT technology, these results better describe the scalability of our parallel algorithm. However, comparing the times T_p for the same number of used nodes, we clearly see that the use of SMT mode is indispensable for our application and it is very efficient.

4. Conclusions

Summarizing, we note that parallelization of our mixed-stable modelling algorithm at the maximum likelihood target function level is very efficient. Some of the obtained efficiency numbers are close to 100% , for example, see Table 2. This means that the time spent in other operations of optimization method is insignificant. Hence the parallelization at this level is fully sufficient and there is no need in the parallelization of algorithmic steps of optimization method.

We note that our parallel algorithm is highly scalable. Speedups up to 800 times were obtained for 1024 parallel processes (see Table 3). The properties of parallel algorithm (the ratio between calculations and data movement) are highly suitable not only for the distributed, but also for shared memory systems (e.g. multicore).

Moreover, our parallel application is able to efficiently utilize the Simultaneous multithreading (Intel Hyper-Threading) technology in modern processors. The obtained performance gains are as significant as one might expect from the doubling of the number of physical cores. Furthermore, the properties of our parallel algorithm look very promising for the application of GPU computing [5].

Finally, the processing of all 29 DAX stocks returns series with the 10-second time step takes too long to be computed in a reasonable time with the single process (in serial mode). It takes 6403.56 seconds with 64 processes and 442.34 seconds with 1024 processes in Simultaneous multithreading mode.

This research shows that the application of modern parallel technologies allows a fast and accurate maximum likelihood estimation of mixed-stable parameters even for large amounts of data. This supports the employment of stable and mixed-stable modelling in the statistical data processing, in particular in high-frequency financial data analysis.

Acknowledgment

The work has been performed under the Project HPC-EUROPA2 (Project number 1020), with the support of the European Community - under the FP7 "Research Infrastructures" Programme.

References

[1] **I. Belovas, A. Kabašinskas, L. Sakalauskas**. A study of stable models of stock markets. *Information Technology and Control*, 2006, Vol. 35, No. 1, 34–46.

[2] **I. Belov, V. Starikovičius**. Parallelization of α-stable modelling algorithms. *Mathematical Modelling and Analysis*, 2007, Vol. 12, No. 4, 409–418.

[3] **J. Diamond, M. Burtscher, J. McCalpin, K. Byoung-Do, S. Keckler, J. Browne.** Evaluation and optimization of multicore performance bottlenecks in supercomputing applications. In: *Proceedings of 2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2011, pp. 32–43.

[4] General-purpose computing on graphics processing units. *www.gpgpu.org*.

[5] **A. Grothey.** Financial Applications: Parallel Portfolio Optimization. *Parallel computing. Eds. R. Trobec, M. Vajteršic, P. Zinterhof. Springer, London*, 2009, 435-469.

[6] **W. Hwu.** GPU Computing Gems. *Elsevier*, 2012 (Section 5. Computational finance).

[7] **A. Igumenov, T. Petkus.** Analysis of Parallel Calculations in Computer Network. *Information Technology and Control*, 2008, Vol. 37, No. 1, 57–62.

[8] **A. Kabašinskas, S. Rachev, L. Sakalauskas, W. Sun, I. Belovas.** Alpha-stable paradigm in financial markets. *Journal of Computational Analysis and Applications*, 2009, Vol. 11, No. 3, 642–688.

[9] **L. Kaklauskas.** Fraktalinių procesų kompiuterių tinkluose stebėsenos ir valdymo metodų tyrimas. *PhD thesis, Vilnius University Institute of Mathematics and Informatics*, 2012.

[10] **G. Kappel.** High Performance Computing in Finance. On the Parallel Implementation of Pricing and Optimization Models. *PhD thesis, Vienna University of Technology*, 2006.

[11] **A. Kronrod.** Nodes and weights of quadrature formulas. Sixteen-place tables. *New York: Consultants Bureau*, 1965.

[12] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. www.mpi-forum.org, Version 1.1, 1995.

[13] **J. Nelder, R. Mead.** A simplex method for function minimization. *Computer Journal*, 1965, Vol. 7, 308–313.

[14] **D. Ojeda.** Comparative study of stable parameter estimators and regression with stably distributed errors. *PhD thesis, American University*, 2001.

[15] OpenMP Group. *www.openmp.org*, 2001.

[16] **S. Rachev, S. Mittnik.** Stable Paretian Models in Finance. *John Wiley and Sons, New York*, 2002.

[17] **V. Starikovičius, R. Čiegis, O. Iliev.** A parallel solver for the design of oil filters. *Mathematical Modelling and Analysis*, 2011, Vol. 16, No. 2, 326–341.

[18] **G. Samorodnitsky, M. Taqqu.** Stable Non-Gaussian Random Processes: Stochastic Models with Infinite Variance. *Chapman & Hall, New York-London*, 2000.

[19] **V. Surkov.** Parallel Option Pricing with Fourier Space Time-Stepping Method on Graphics Processing Units. *Parallel Computing*, 2010, Vol. 36, No. 7, 372–380.

[20] **W. Wasserfallen, H. Zimmermann.** The behavior of intradaily exchange rates. *Journal of Banking and Finance*, 1985, Vol. 9, 55–72.

Received June 2014.