Vestnik KRAUNC. Fiz.-Mat. nauki. 2023. vol. 43. no. 2. P. 87-110. ISSN 2079-6641

INFORMATION AND COMPUTATION TECHNOLOGIES

https://doi.org/10.26117/2079-6641-2023-43-2-87-110

Research Article

Full text in English

MSC 34A08, 65Y05, 65M06

Parallelization of a Numerical Algorithm for Solving the Cauchy Problem for a Nonlinear Differential Equation of Fractional Variable Order Using OpenMP Technology

D. A. Tverdyi[1,2], R. I. Parovik[1,2*], A. R. Hayotov[3], A. K Boltaev[3]

[1] Vitus Bering Kamchatka State University,
Russia, 683032, Petropavlovsk-Kamchatsky, Pogranichnaya st., 4.

[2] Institute for Cosmophysical Research and Radio Wave Propagation FEB RAS,
Russia, 684034, Paratunka, Mirnaya st., 7.

[3] V.I. Romanovskiy Institute of Mathematics, Uzbekistan Academy of Sciences,
100174, Tashkent, University st., 9.

Abstract. The article presents a software implementation of a parallel efficient and fast computational algorithm for solving the Cauchy problem for a nonlinear differential equation of a fractional variable order. The computational algorithm is based on a non-local explicit finite-difference scheme, taking into account the approximation of the Gerasimov-Caputo fractional derivative VO included in the main differential equation. The algorithms for parallelization of the non-local explicit finite difference scheme were implemented as functions of the user library of the *C* programming language using the OpenMP technology. The OpenMP technology allows implementing parallel algorithms for working with the CPU computing node using its multithreading. The *C* language was chosen because of its versatility and lack of strict restrictions on memory handling. Further in the paper, the efficiency of the parallel algorithm is investigated. Efficiency is understood as the optimal ratio in coordinates: acceleration of calculations – the amount of RAM memory occupied, in comparison with the sequential version of the algorithm. The average computation time is analyzed in terms of: running time, acceleration, efficiency and cost of the algorithm. These algorithms were run on two different computing systems: a gaming laptop and a computing server. For a non-local explicit scheme, a significant performance increase of 3-5 times is shown for various methods of software implementation.

Key words: fractional derivatives, heredity, memory effect, finite difference schemes, parallel computing, OpenMP

Received: 02.06.2023; Revised: 09.06.2023; Accepted: 20.06.2023; First online: 30.06.2023

For citation. Tverdyi D. A., Parovik R. I., Hayotov A. R., Boltaev A. K. Parallelization of a numerical algorithm for solving the Cauchy problem for a nonlinear differential equation of fractional variable order using OpenMP technology. *Vestnik KRAUNC. Fiz.-mat. nauki.* 2023, **43**: 2, 87-110. EDN: WFDGQO. https://doi.org/10.26117/2079-6641-2023-43-2-87-110.

Funding. This research was funded by grant of the President of the Russian Federation grant number MD-758.2022.1.1 on the topic "Development of mathematical models of fractional dynamics in order to study oscillatory processes and processes with saturation".

Competing interests. There are no conflicts of interest regarding authorship and publication.

Contribution and Responsibility. All authors contributed to this article. Authors are solely responsible for providing the final version of the article in print. The final version of the manuscript was approved by all authors.

***Correspondence:** E-mail: romanparovik@gmail.com

Вестник КРАУНЦ. Физ.-мат. науки. 2023. Т. 43. №2. С. 87-110. ISSN 2079-6641

ИНФОРМАЦИОННЫЕ И ВЫЧИСЛИТЕЛЬНЫЕ ТЕХНОЛОГИИ

https://doi.org/10.26117/2079-6641-2023-43-2-87-110

Научная статья

Полный текст на английском языке

УДК 519.642.2, 519.687.1

Распараллеливание численного алгоритма решения задачи Коши для нелинейного дифференциального уравнения дробного переменного порядка с помощью технологии OpenMP

***Д. А. Твёрдый*[1,2], *Р. И. Паровик*[1,2*], *А. Р. Хаётов*[3], *А. К. Болтаев*[3]**

[1] ФГБОУ ВО Камчатский государственный университет им. Витуса Беринга, Россия, 683032, г. Петропавловск-Камчатский, ул. Пограничная, 4.

[2] ФГБУН Институт космофизических исследований и распространения радиоволн ДВО РАН, Россия, 684034, с. Паратунка, ул.Мирная, д. 7.

[3] Институт математики имени В. И. Романовского Академии наук Республики Узбекистан, Узбекистан, 100174, г. Ташкент, ул. Университетская, д. 9.

Аннотация. В статье представлена программная реализация параллельного эффективного и быстрого вычислительного алгоритма решения задачи Коши для нелинейного дифференциального уравнения дробного переменного порядка. Вычислительный алгоритм основан на нелокальной явной конечно-разностной схеме с учетом аппроксимации дробной производной VO Герасимова-Капуто, входящей в основное дифференциальное уравнение. Алгоритмы распараллеливания нелокальной явной конечно-разностной схемы были реализованы в виде функций пользовательской библиотеки языка программирования *C* с использованием технологии OpenMP. Технология OpenMP позволяет реализовывать параллельные алгоритмы для работы с вычислительным узлом CPU, используя его многопоточность. Язык *C* выбран из-за его универсальности и отсутствия в нем строгих ограничений при работе с памятью. Далее в работе исследуется эффективность параллельного алгоритма. Под эффективностью понимается оптимальное соотношение в координатах: ускорение вычислений – объём занимаемой RAM памяти, по сравнению с последовательной версией алгоритма. Анализируется среднее время вычисления в терминах: время работы, ускорение, эффективность и стоимость алгоритма. Данные алгоритмы были запущены на двух различных вычислительных системах: игровом ноутбуке и вычислительном сервере. Для нелокальной явной схемы показан существенный прирост производительности в 3-5 раз при различных методах программной реализации.

Ключевые слова: дробные производные, эредитарность, эффект памяти, явные конечно-разностные схемы, параллельные вычисления, OpenMP

Получение: 02.06.2023; Исправление: 09.06.2023; Принятие: 20.06.2023; Публикация онлайн: 30.06.2023

Для цитирования. Tverdyi D. A., Parovik R. I., Hayotov A. R., Boltaev A. K. Parallelization of a numerical algorithm for solving the Cauchy problem for a nonlinear differential equation of fractional variable order using OpenMP technology // *Вестник КРАУНЦ. Физ.-мат. науки.* 2023. Т. 43. № 2. С. 87-110. EDN: WFDGQO. https://doi.org/10.26117/2079-6641-2023-43-2-87-110.

Финансирование. Исследования выполнены рамках гранта Президента РФ МД-758.2022.1.1 по теме «Развитие математических моделей дробной динамики с целью исследования колебательных процессов и процессов с насыщением».

Конкурирующие интересы. Конфликтов интересов в отношении авторства и публикации нет.

Авторский вклад и ответственность. Авторы участвовали в написании статьи и полностью несут ответственность за предоставление окончательной версии статьи в печать.

*__Корреспонденция:__ E-mail: romanparovik@gmail.com

Introduction

As many scientific studies around the world show, equations with fractional derivatives and fractional integrals are increasingly finding their practical applications in a variety of problems: in meteorology and diffusion [1], chemistry [2], used in modeling the movement of fluid flows [3] and propagation of complex acoustic vibrations [4] and at the cutting edge of science – quantum physics [5].

Equations arising during the construction of mathematical models for describing the processes of fractional and fractional-nonlinear dynamics can be solved using the methods of numerical schemes [6], which have a higher computational load than equations in ordinary derivatives. As a rule, the complication of the numerical scheme occurs due to the introduction of nontrivial discrete analogues (approximations) of fractional differentiation and integration operators. From here, there is a need to build parallel algorithms for solving fractional dynamics models, the main difficulty of which is to develop effective methods for shooting approximations of the used fractional operator. For example, in the works of V. Bohaienko [7], a model of the process of salt transport in fractally structured media [8] is considered based on equations with Gerasimov-Caputo fractional derivatives [9, 10] with respect to spatial variables, for which edge task. For the numerical solution, a locally one-dimensional finite-difference scheme is derived. Parallel algorithms are proposed for systems with distributed memory and GPU [11]. Reduction of computational complexity is achieved due to the proposed new procedure for approximating the Gerasimov-Caputo fractional derivative. Further ideas for developing parallel algorithms for GPUs were developed in [12], where a finite-difference numerical scheme for solving a three-dimensional model diffusion problem based on the ϕ–Caputo derivative is implemented. It was shown that when using the approach based on the expansion of the kernel of the integral operator into series, it is possible to obtain an acceleration of calculations by an order of magnitude.

As is known, the use of the fractional calculus apparatus allows one to take into account in the mathematical model non-locality both in time and space, using one or another fractional operator from a large number of their definitions [13], both of constant order and VO [14, 15] . However, it should be borne in mind that this further increases the computational load in the model problem, and also reduces the potential for parallelization of the implemented numerical solution schemes, due to the need to take into account the previous states of the simulated process.

In this article, to solve the Cauchy problem for a non-linear differential equation with a fractional derivative VO Gerasimov-Caputo of fractional variable order [16] using a non-local explicit finite difference scheme (EFDS), we present an efficient software parallel implementation based on OpenMP technology using CPU [17].

One of the differences between this article and those considered in the review is that for the model equation we consider a one-dimensional initial problem local in space, but nonlocal in time [18]. The main difference between the implementation of the parallel algorithm proposed in this article and the algorithms from the review of scientific papers is that the model equation considers a generalization of the Gerasimov-Caputo operator for the case of a variable order $\alpha(t)$ - non-constant in time, and its

approximations are introduced according to [19,20]. Variable nonlocality in time can be applicable for a more accurate and correct description of some dynamic processes by mathematical models [21].

It can be seen from the literature review that there are no studies devoted to the development of parallel algorithms for solving the Cauchy problem for a non-linear fractional equation with Gerasimov-Caputo VO, in particular, the parallelization of an explicit finite-difference scheme for solving the fractional Riccati equation.

The research plan of the article is as follows: Section describes the hardware basis for which the proposed efficient parallel numerical solution algorithm has been developed; Section gives a definition of the variable order fractional derivative used in the mathematical formulation of the problem; Section describes the Cauchy problem for a non-linear equation with a fractional derivative VO Gerasimov-Caputo. This statement of the problem is the basis for constructing mathematical models of various dynamic processes. In particular, the fractional Riccati equation is considered; Section presents a non-local explicit finite difference scheme, using the fractional Riccati equation as an example, for solving direct problems of fractional dynamics; Section , in the form of code blocks of the C programming language with elements of OpenMP, describes a parallel algorithm that implements an explicit numerical scheme. All code located in blocks is a complete function that implements a numerical scheme; Section analyzes the performance of the presented parallel explicit finite difference scheme algorithm in comparison with its best sequential version.

Hardware basis

It's hard to overestimate how deeply computers have penetrated every aspect of our lives, from simple devices like the toaster to the latest spacecraft. The basis of any computer is the corresponding central processing unit (CPU), which is responsible for executing program instructions and coordinating all other vital nodes of the computing system [22]. The core in the CPU is a set of control and logic blocks, and most importantly, arithmetic blocks (ALU), as well as the necessary data buses for their transmission, and several levels of caches for storing and accessing data as quickly as possible. Thus, the Core is an autonomous microprocessor, the set of blocks of which allows you to independently process program instructions, and capable of executing a computer program.

An important technical aspect of the CPU is the distinction between:

- Physical cores – specific hardware blocks implemented on transistors and other circuit parts that make up the core [22];
- Logical cores – independent threads to which the CPU is able to distribute the execution of instructions.

This separation is made possible by factors more dependent on the operating system (OS) managing processes and threads than on the CPU itself.

Remark 1. Then, on such a computing node with multithreading, it is possible to execute a program specially designed for this purpose - a parallel algorithm.

For almost 40 years, one of the important methods of improving the performance of computing systems has been to increase the clock frequency of the CPU, by miniaturizing transistors and increasing their number. This led to an increase in power consumption in proportion to the frequency, and hence to an increase in current leakage and subsequent temperature damage to the chip. This phenomenon is also called the "energy wall one of the three fundamental problems of CPU design development. As a result, transistor manufacturers were forced to look for alternatives and rethink the CPU device in order to circumvent this limitation.

Almost immediately, the developers came up with the idea that it is possible to combine more than one physical CPU on one motherboard, but with a common RAM (Random Access Memory) memory and access to the rest of the peripherals. Like most system innovations, multiprocessor systems were usually developed for specific tasks, which often came from the scientific and military-defense environment. For example, problems of mathematical physics, where numerical schemes are used to solve equations, and for the correct solution of such a scheme, sometimes a very small sampling step is needed, that is, calculations are carried out very many times. Sometimes numerical schemes can be divided into several independent parts that can be solved in parallel, but at some stage their results will need to be coordinated.

Remark 2. On a system with one physical core, and even more so with one thread, the execution of such algorithms will take an extremely long time, days and months! If we want to get a solution in a shorter time, we will need ultra-high performance systems. However, many CPUs are forced to interact with each other via RAM and external buses, which means large performance delays. This led the developers to the concept of «multicore» CPUs consisting of more than one core on a chip and functioning as a single unit [24].

If only your PC (personal computer) no older than two decades, then most likely the CPU in it is multicore. Since the cores of a multi-core CPU are placed close on one chip, the data transfer rate between the cores of such a CPU is much faster. Moreover, these cores, in addition to their own caches, have shared caches, which improves and significantly speeds up inter-core communication. In addition, unlike multiprocessor systems, the degree of interaction of cores and their consistency makes it possible to better adapt the performance of a multi-core CPU to consumer tasks for which there is enough capacity of consumer PCs.

One of the disadvantages is that increasing the number of cores never gives an ideal increase in performance. The performance of any parallel program is limited by a part of the code that cannot be parallelized. This limitation is another «wall of parallelism» in the development of the CPU design, also called Amdahl's law [25].

Remark 3. The hardware and software limitation of the efficiency of acceleration from parallelization, when using a large number of CPU cores, is the high time spent on parallelizing the task and assembling all the results of calculations back. What we will show in the analysis of the developed effective algorithms running on a super computer

with a large number of threads. This leads to difficulties in efficiently loading a large number of cores.

Currently, supercomputer technologies [26, 27] and modeling of various phenomena on ultra-high performance systems, such as supercomputers and servers, have been widely developed. In the tasks we are considering, over time, it also becomes necessary to use a super computer and its computing power. The first and obvious reason is the ability to conduct modeling with an extremely small h sampling step, due to the amount of RAM memory in which it is possible to operate with matrices and vectors with huge dimensions. This allows us, when developing algorithms that implement numerical schemes for solving direct problems, to concentrate on the speed of operation and optimal loading of CPU cores.

The second reason will be that the presented algorithms for solving direct problems will be used in solving inverse problems. With the help of which we plan to restore the values of coefficients and parameters of the model of a certain physical process. The difficulty is that some methods of implementing inverse problems require solving a direct problem at each such h discretization step. A super computer, in turn, will allow you to perform such algorithms much faster.

Fractional derivative VO Gerasimov-Caputo

There are many tens of definitions of the fractional differentiation operator, but we will focus on using one of them [13], namely the Gerasimov-Caputo fractional derivative [9, 10]. According to [19], the Gerasimov-Caputo derivative generalizes to the case of a fractional variable order $(0;1)$ as follows:

Definition 1. Gerasimov-Caputo variable-order fractional differentiation operator [19] (or VO [14, 15] Gerasimov-Caputo), where $0 < \alpha(t) < 1$ – non-constant order of the fractional derivative, $u(t) \in C[0,T]$, are called:

$$\partial_{0t}^{\alpha(t)} u(\sigma) = \frac{1}{\Gamma(1-\alpha(t))} \int_0^t \frac{\dot{u}(\sigma)}{(t-\sigma)^{\alpha(t)}} d\sigma, \tag{1}$$

where the derivative $\dot{u}(t) = \frac{du}{dt}$, and $t \in [0,T]$ is the current simulation time, $T > 0$ is the total simulation time, and $\Gamma(.)$ – Euler's gamma function is:

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt, \qquad x \in \mathbb{C} : \Re(x) > 0, \tag{2}$$

where (2) is monotonically decreasing on the interval $0 < x < 1$.

Remark 4. Note that there is a definition of the Gerasimov-Caputo fractional derivative VO (1) for the case when $1 < \alpha(t) < 2$ according to [28], and it is also possible to generalize the fractional derivative VO Gerasimov-Caputo for the case of $n < \alpha(t) < n+1$, according to the work [29].

The concept of heredity (memory) was proposed and studied by the Italian mathematician Vito Volterra [30]. In [31], various physical processes with heredity effects were studied. We will not dwell on the theoretical aspects related to the description of

the memory effect (heredity) using fractional derivatives. You can learn more about this in [19, 20].

Remark 5. It is important to note that according to the principles of hereditarity introduced by V. Volterra [30], the dynamic process is studied on the interval $(-\infty, t)$. That is, it is necessary to take into account the entire history of the process, and this is impossible. In our studies, hereditarity is determined on the interval $(0, t)$, which means that only the history of the process that took place during the experiment, which began from the time moment $t = 0$, is taken into account.

Mathematical statement of the problem

Consider for the following non-linear differential equation of fractional variable order with non-constant coefficients, the following Cauchy problem:

$$\partial_{0t}^{\alpha(t)} u(\sigma) = b(t)u(t) + f(u(t), t), \qquad u(0) = u_0, \tag{3}$$

where, $f(u(t), t)$ is a nonlinear function satisfying the Lipschitz condition with constant $L > 0$ in the variable $u(t)$: $|f(u_1, t) - f(u_2, t)| < L|u_1 - u_2|$; $\partial_{0t}^{\alpha(t)} u(\sigma)$ – fractional differentiation operator in the sense of (1); $u(t) \in C^2[0, T]$ – unknown solution function; u_0 – given constant, initial condition of the Cauchy problem. Cauchy problem (3) describes a wide class of dynamic processes with variable memory in environments with heredity [21, 29, 32].

Consider a more specific form of the equation (3) by taking $f(u(t), t) = -a(t)u(t)^2 + c(t)a$ $a(t), b(t)$ $c(t) \in C[0, T]$ are given functions that determine the values of the coefficients at each moment of time. Let other task parameters be similar (3). Then we get a fractional analogue of the well-known Riccati equation [33]:

$$\partial_{0t}^{\alpha(t)} u(\sigma) + a(t)u^2(t) - b(t)u(t) - c(t) = 0, \qquad u(0) = u_0. \tag{4}$$

Non-local explicit finite difference scheme

To solve the problem (3), we will resort to the method of numerical solution of differential equations. One of the methods implemented in this manuscript will be a relatively simple nonlocal explicit Finite-difference scheme (EFDS) (from the English nonlocal Explicit Finite-Difference scheme) compiled by the Euler method [34]. However, there is a difficulty associated with the need to obtain a discrete analogue (approximation) of the Gerasimov-Caputo fractional derivative VO (1).

To obtain a numerical solution using finite difference schemes [19, 35], consider (3) on a uniform grid with $h = T/N$ – sampling step, where N is the number of grid nodes.

According to the Euler method , it can be represented as:

$$
\begin{gathered}
u_1 = \frac{1}{A_0}\left(\left(A_0(1-w_1^0)+b_0\right)u_0 - a_0u_0^2 + c_0\right),\\
u_2 = \frac{1}{A_1}\left(\left(A_1\left(1-w_1^1\right)+b_1\right)u_1 + A_1w_1^1u_0 - a_1u_1^2 + c_1\right),\\
u_{i+1} = \frac{1}{A_i}\left(\left(A_i\left(1-w_1^i\right)+b_i\right)u_i + A_iw_1^iu_{i-1} - \right.\\
\left. A_i\sum_{j=2}^{i} w_j^i\left(u_{i-j+1}-u_{i-j}\right) - a_iu_i^2 + b_iu_i + c_i\right),\\
A_i = \frac{h^{-\alpha_{i+1}}}{\Gamma(2-\alpha_{i+1})}, \qquad w_j^i = (j+1)^{1-\alpha_{i+1}} - j^{1-\alpha_{i+1}},\\
i = 2,\ldots,N-1, \qquad u_0 = C,\\
w_1^i = \sum_{i=0}^{N-1}\left(2^{1-\alpha_{i+1}} - 1^{1-\alpha_{i+1}}\right),
\end{gathered}
\tag{5}
$$

where C is a known constant.

In [19, p.14], questions of convergence and stability of EFDS (5) are investigated. It is shown that the nonlocal explicit scheme is stable under the condition:

$$
h \le \frac{2^{1-\max\limits_i(\alpha_i)}-1}{\max_i(b_i)} \tag{6}
$$

and converge with the first order of accuracy of $O(h)$.

Remark 6. In this article, the numerical scheme (5) differs somewhat from that presented in [19, p.14], in that a weighting factor w_1^i appears in (5) derived from the coefficient w_j^i to speed up calculations. This does not affect the accuracy of EFDS (5) calculations, nor its convergence and stability.

Software implementation of the EFDS algorithm using OpenMP

To solve the problems of developing parallel numerical algorithms, one of the technologies chosen is OpenMP - an open standard of the software interface for parallel systems with shared memory [17]. It is a set of directives for compilers, libraries of functions and environment variables. OpenMP implements parallel computing using the idea of multithreading – multiple parallel tasks performed by CPU threads.

Remark 7. OpenMP is officially supported by the C, C++ and Fortran programming languages, but implementations can be found for some other languages, such as Pascal and Java. The high-level C language was chosen as the main programming language, for the following reasons:

1. of its versatility and rather large freedom of working with memory [36], which is useful for creating efficient algorithms;

2. the ability to execute OS terminal system commands inside the C code, since the C language was originally developed as a system programming language.

Let's take a step-by-step effective software implementation of EFDS (5) to solve the Cauchy problem (4). We will provide a listing of the code implementing EFDS, divided into parts, for the necessary explanations. Code fragments can be sequentially assembled into a working function implementing EFDS. But first you need to define several data types and auxiliary functions. Data type `MATC` – information about location and time measurements:

```
typedef struct mesure_all_times_calc {
    float  time_calc_res_m_3;
    struct timeval tv1, tv2, dtv;
        char   location[CHAR_LEN];  // where fields MATC object was filled
} MATC;
MATC measure_PARA;  // FOR mesure time working Parallel part code
MATC measure_SEQU;  // FOR mesure time working Sequenitial part code
MATC measure_MAIN;  // FOR mesure time working All part's code
```

Function `tic()` – to start measuring time in objects `MATC`:

```
MATC tic( MATC measure ) {
    gettimeofday( &measure.tv1, NULL );   // most accuracy method !
        return measure;
}
```

Function `toc()` – completion of measurements and calculation of elapsed time:

```
MATC toc( MATC measure ) {
    gettimeofday( &measure.tv2, NULL );
    measure.dtv.tv_sec  = measure.tv2.tv_sec  - measure.tv1.tv_sec;
    measure.dtv.tv_usec = measure.tv2.tv_usec - measure.tv1.tv_usec;
    if( measure.dtv.tv_usec < 0 ) {
        measure.dtv.tv_sec--;
        measure.dtv.tv_usec += 1000000;
    }
    float elapsed_dtv = measure.dtv.tv_sec * 1000 +
                        measure.dtv.tv_usec / 1000;
    measure.time_calc_res_m_3 = (double)elapsed_dtv / 1000;
    return measure;
}
```

Directives defining the values of the parameters of the Cauchy model problem (4) in the numerical scheme ERDS (5):

```
    #define CHAR_LEN 256    // num char elements *char[] massive
    #define EPSILON 1e-2    // numerical methods accuracy
```

```
#define par_alpha(i,h,T) (float)(0.9-((0.1*i*h)/T)
#define coeff_a(i,h,T) (float)(pow(cos(i*h),2)/T)
#define coeff_b(i,h,T) (float)(1-((0.1*i*h)/T))
#define coeff_c(i,h,T) (float)(pow(sin(i*h),2)/T)
```

in the code above already, the functions defining the test example from Chapter .

Parallel EFDS implementation code

Calculation of the EFDS algorithm using OpenMP is performed by the function `EFDS_parallel _OpenMP()`, whose only argument is an object of the `INP_PAR` class, which is declared and filled in using an auxiliary function that reads the necessary parameters from `.txt` user file.

```
OUT_RES EFDS_parallel_OpenMP( INP_PAR input ) {
    memcpy( input.current_method, (char *)"EFDS", CHAR_LEN );
```

This function will be launched within a small software package, all the code of which we do not present in this article, and some of its auxiliary functions are given above. Data type `IMP_PAR`– modeling input parameters.

The allocation of memory for the variables necessary in the calculation process is carried out at the beginning of the function body:

```
int   T = input.T;   //  time of numerical experement.
int   N = input.N;   // number cells of grid (lattice).
float h = (float)T / (float)N;  // step of discretisation.
float u_0 = input.u_0;     // start point for Koshi poblem
OUT_RES output;
int N_p1 = N + 1;
float     a[ N ];    // massive value 'a(t)'.
float     b[ N ];    // massive value 'b(t)'.
float     c[ N ];    // massive value 'c(t)'.
float alpha[ N_p1 ]; // massive value 'alpha(t)'
float     A[ N ];    // 1-st Weight coefficient 'A'.( for EFDS )
float   w_1[ N ];    // special value 'w' for (j = 1) and i = 0 .. N.
float     u[ N_p1 ]; // solve. ( + 1 because u[0] include)
float summ_diff_u;   // contain summ difference value 'u'
//  allocated memory for 'w' like -- "lower triangular matrix"
float **w;
w = (float**) malloc( N * sizeof(float*) );
for( int i = 0; i <= N - 1; i++ ) {
    w[i] = (float*) malloc((i + 1) * sizeof(float));
}
```

At the end of this block of code, we also count the amount of memory we have occupied for the most heavy arrays. This will help us determine the scale of the mathematical

model, determined by the sampling step $h = T/N$. Since the dimensions of vectors and matrices, the main consumers of RAM memory, depend on the N parameter.

```
float mem_aloc_CPU = sizeof u +sizeof a + sizeof b + sizeof c +
                     sizeof alpha + sizeof A + sizeof w_1;
mem_aloc_CPU += (( (float)N * (float)N )/2 * 4); // sizeof w
```

Next, the `tic()` there is a starting time measurement in the previously created objects of the `MATC` structure, which will be spent by the entire algorithm and its individual parts.

```
// START mesure time working only Parallel code
measure_PARA = tic( measure_PARA );
// START mesure time working All numerical method
measure_MAIN = measure_PARA;
```

Next comes the computational grid initialization block. In other words, we tell the system «which hardware resources we can use for useful calculations». Since we are working with OpenMP here we will set the number of CPU threads to execute the parallel code:

```
printf( "\n    ---- Initialize calculation grid ----\n" );
printf( "\n    OpenMP : " );
omp_set_num_threads( input.num_threads_CPU );
#pragma omp parallel  {
    printf( "\n       omp_get_num_threads() = %d",
    omp_get_num_threads() );
}
printf( "\n    ---- grid done. ----\n\n" );
```

Now start implementing blocks of parallel code, where the 1st block looks like this:

```
printf( "    START : OpenMP : 1 : (threads - %d) ",
        input.num_threads_CPU);
#pragma omp parallel shared( a, b, c, alpha ) {
    #pragma omp for schedule( static ) nowait
    for( int i = 0; i <= N - 1; ++i )  {
        a[i]   =   coeff_a(i,h,T);
        b[i]   =   coeff_b(i,h,T);
        c[i]   =   coeff_c(i,h,T);
    }
    #pragma omp for schedule( static ) nowait
    for( int i = 0; i <= N; ++i )  {
        alpha[i] = par_alpha(i,h,T);
    }
}
printf( "   ...    sucsessfull END \n" );
```

here for EFDS (5) the discrete values of the coefficients of the equation are calculated, at each i node of the grid of N nodes. The formulas by which the values of the coefficients of the equation in (5) are calculated are set by functions, or rather by the directives described earlier. The calculation of N values from $i = 0$ to $i = N$ is spread over multiple threads. The number of values computed by each of the threads, i.e. computational load, set uniform, by means of the directive `schedule()` with the `static` argument.

In the 2nd block of the parallel code, the weight coefficients A_i and w_1^i for EFDS (5) defined in memory as arrays are calculated using reduction: `A[i]` and `w_1[i]`:

```
printf( "\n    START : OpenMP : 2 : (threads - %d) ",
        input.num_threads_CPU);
for( int i = 0; i <= N - 1; ++i )  {
    w_1[i] = 0.0;
}
#pragma omp parallel shared(A) reduction ( +: w_1 )  {
    #pragma omp for schedule( static ) nowait
    for( int i = 0; i <= N - 1; ++i )  {
        A[i] = (pow( h, - alpha[i + 1] )) /
               (exp( lgamma( 2 - alpha[i + 1] ) ));
    }
    #pragma omp for schedule( static ) nowait
    for( int i = 0; i <= N - 1; ++i )  {
        w_1[i] += pow( 2, 1 - alpha[i + 1] ) -
                  pow( 1, 1 - alpha[i + 1] );
    }
}
printf( "   ...    sucsessfull END \n" );
```

Remark 8. The formula A_i contains the Euler gamma function (2). In the standard library of mathematical functions <math.h> of the C language, there is no corresponding function (2), but there is `lgamma()` – the logarithm of the gamma function. Then for the desired result you need to take `exp(lgamma())` as shown above.

Calculation of the 3rd EFDS weighting factor from (5), which is in memory `w[i][j]` – a lower-triangular matrix:

```
printf( "\n    START : OpenMP : 3 : (threads - %d) ",
        input.num_threads_CPU );
#pragma omp parallel shared( w )  {
    // Create: "lower triangular matrix" (i >= j or j <= i) matrix
    //  (i - strings, j - columns) and contain's value.
    #pragma omp for schedule( static ) nowait
    for( int i = 0; i <= N - 1; i++ )  {
        for( int j = 0; j <= i; j++ )  {
         w[i][j] = pow( j + 1, 1 - alpha[i + 1] ) -
                   pow(     j, 1 - alpha[i + 1] );
```

```
        }
    }
}
printf( "   ...    sucsessfull END \n" );
```

Now we can measure time spent on parallel code in blocks 1-3 using the `toc()` function:

```
// END mesure time working only Parallel code
measure_PARA = toc( measure_PARA );
memcpy( measure_PARA.location ,
        (char *)"{ EFDS _parallel OpenMP_ block }", CHAR_LEN );
// START mesure time working Sequential part code
measure_SEQU = tic( measure_SEQU );
```

and also give a starting measurement of the running time of the serial code.

The sequential part of the EFDS algorithm (5) is an unavoidable part for our task since there is a dependency in the solution function: the value of `u` in the node `u[i+1]` requires knowledge of the value in the previous node `u[i]`. Such a part of the algorithm cannot be distributed by definition, and this cannot be avoided when implementing explicit finite-difference numerical schemes.

```
printf( "\n    START : Sequential : ");
u[0] = u_0;
u[1] = ( ( A[0] * (1 - w_1[0]) + b[0] ) *
           u[0] - a[0] * pow( u[0], 2 ) + c[0] ) / A[0];
u[2] = ( ( A[1] * (1 - w_1[1]) + b[1] ) *
           u[1] + A[1] * w_1[1] * u[0] -
           a[1] * pow( u[1], 2 ) + c[1] ) / A[1];
// MAIN cycle.
for( int i = 2; i <= N - 1; ++i )  {
    summ_diff_u = 0;
    for( int j = 2; j <= i; ++j )  {
     summ_diff_u += w[i][j] * (u[i - j + 1] - u[i - j]);
    }
    u[i+1] = ( ( A[i] * (1 - w_1[i]) + b[i] ) *
                 u[i] + A[i] * w_1[i] * u[i-1] -
               ( A[i] * summ_diff_u ) -
                 a[i] * pow( u[i], 2 ) + c[i] ) / A[i];
}
printf( "     ...    sucsessfull END \n" );
```

Final measurements of the calculation time:

```
// END mesure time working Sequential part code
measure_SEQU = toc( measure_SEQU );
memcpy( measure_SEQU.location,
```

```
        (char *)"{ EFDS _sequential_ block }", CHAR_LEN );
// END mesure time working All numerical method
measure_MAIN = toc( measure_MAIN );
memcpy( measure_MAIN.location,
        (char *)"{ EFDS _all_ algorihm }", CHAR_LEN );
```

Let's check the fulfillment of the stability condition (6) described in the auxiliary function `criterion_stability()`. After that, the execution of the EFDS algorithm (5) can be considered completed.

```
    criterion_stability( (char *)"EFDS", T, N, h, a, b, c, alpha );
    printf( "\n End calc               %s. \n", input.current_method );
```

At the end of the `EFDS_parallel _OpenMP()` free the memory used for storing the values of the lower triangular matrix `w[i][j]` weighting coefficient:

```
    for( int i = 0; i <= N - 1; i++ )
        free(w[i]);
    free( w );
```

At the end, return the result of the function to the `OUT_RES` object, where data type `OUT_RES` – simulation results and output parameters:

```
    memcpy(    output.type_method,    input.type_method, CHAR_LEN );
    memcpy( output.type_algorithm, input.type_algorithm, CHAR_LEN );
    output.num_threads_CPU = input.num_threads_CPU;
    output.mem_aloc_CPU = mem_aloc_CPU;
    output.T   = T;
    output.N   = N;
    output.u_0 = u[0];
    output.h   = h;
    output.res = (float*) malloc( N_p1 * sizeof(float) );
    for( int i = 0; i <= N ; i++ )
        output.res[i] = u[i];
    return output;
} % END FUNCTION
```

Analysis of computational efficiency

Let's investigate how much faster the algorithm runs with the parameter N – characterizing the complexity of the algorithm. Comparing the computation time of the scheme (5) on: one processor thread (serial algorithm) and multiple threads (parallel algorithm). We will conduct experiments on the following computing systems with different numbers of threads.

The first system will be the *HP Pavilion Gaming Laptop Z270X 15-dk0xxx* used for development, debugging, and computational experiments. The laptop has

the following specifications: CPU: Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz (4 CPUs, 8 Threads); RAM: 8192 Mb GPU: NVIDIA GeForce GTX 1650, GPU @ 16 Multiprocessors 1.56GHz (1024 CUDA cores), 4102 Mb.

The second system is a personal supercomputer *NVIDIA DGX STATION* with one multi-threaded CPU, which we also used for computational experiments. This super computer has the following specifications: CPU: Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz(~1.20GHz) (20 CPUs, 40 Threads); RAM: 263855 Mb GPU$\times$4: NVIDIA Tesla V100-DGXS, GPU @ 80 Multiprocessors 1.53GHz (5120 CUDA cores), 34084 Mb. The personal supercomputer *NVIDIA DGX STATION* at the Institute of Mathematics named after V.I. Romanovsky of the Academy of Sciences of the Republic Uzbekistan.

It follows from Brent's lemma [17, 37, 38] that any algorithm implemented for execution in PRAM type systems can be modified in such a way that it can be executed in PRAM systems with fewer threads. Where PRAM is a parallel random access memory machine that combines c threads (cores or CPUs), shared memory, and a controller. This is one of the most common models of computing systems [17]. This means – we can run parallel algorithm on systems with different numbers of cores or threads.

Remark 9. We will measure the time spent on code execution using the `gettimeofday()` function, which returns `tv\_sec` and `tv\_usec` structures `timeval` seconds and microseconds, respectively, since midnight January 1, 1970. This is one of the most accurate methods for calculating program execution time. The method is implemented in two functions: `tic()` – starts freezing, and `toc()` – stops the measurement and calculates the elapsed time.

Now, using data on the time spent on code execution, we can investigate the efficiency of the parallel algorithm. Efficiency is understood as the optimal ratio in coordinates: acceleration of calculations - the amount of RAM memory occupied, in comparison with the sequential version of the algorithm. The average computation time is analyzed in terms of: running time, acceleration, efficiency and cost of the algorithm.

To measure computation time, consider a model (4) with the following parameters: $\alpha(t) = 0.9 - \frac{0.1t}{T}$; $a(t) = \frac{\cos^2(t)}{T}$; $b(t) = 1 - \frac{0.1t}{T}$; $c(t) = \frac{\sin^2(t)}{T}$; N =30,000; T =20. The result of the EFDS calculation (5) is shown in the Figure 1:

According to [17, p.20], we introduce the following notions:

- let $N = 30000$ be the number of grid nodes in the scheme (5), characterize N the complexity of the algorithm;
- $T_1(N)$ – time taken by the sequential algorithm to solve a problem with N complexity;
- $T_c(N)$ – time taken by the parallel OpenMP algorithm for $c > 1$ threads;

Due to the fact that we will get slightly different T_c for each new numerical experiment, and the number of experiments is finite, then T_c can be considered a random

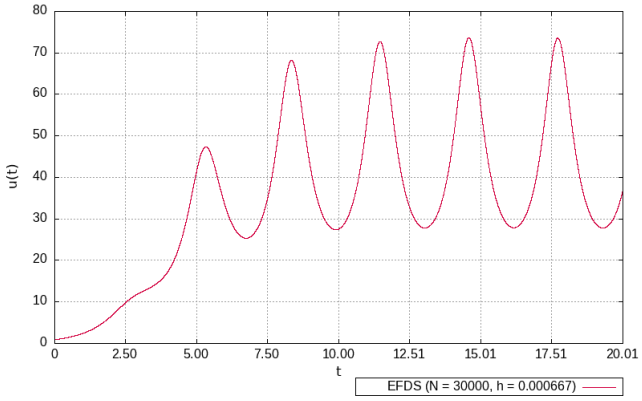


Fig. 1. EFDS example calculation result (5)

variable with some distribution function. Then, to calculate $T_1(N)$ and $T_c(N)$, one can use concept of mathematical expectation (mean value) for a discrete random variable:

$$\mathbb{E}(T_c(N)) = \sum_{i=1}^{L} \frac{T_c^i(N)}{L}, \tag{7}$$

where i is the index of the numerical experiment.

Then for the developed algorithms, from a sample of $L = 10$ values, we determine the average (7) value of $\mathbb{E}(\cdot)$, with a different number of c CPU threads involved. The results for EFDS are presented in the Table 1 and Figure 2 below:

Table 1

Time spent T (sec.) and RAM memory (Mb) for calculating EFDS algorithms based on OpenMP parallelism

EFDS	Notebook		SuperPC				
i	$T_1(N)$	$T_6(N)$	$T_1(N)$	$T_6(N)$	$T_{17}(N)$	$T_{28}(N)$	$T_{39}(N)$
1	19,07	6,96	26,73	10,04	4,885	3,86	3,487
2	19,04	6,36	26,74	10,05	4,887	3,815	3,355
3	19,06	6,818	26,75	10,06	4,91	3,82	3,467
4	19,02	7,259	26,78	10,05	4,883	3,777	3,457
5	19,02	6,754	26,75	10,05	4,885	3,741	3,444
6	19,03	7,006	26,76	10,05	4,884	3,913	3,4
7	19,37	7,085	26,77	10,05	4,913	3,807	3,5
8	19,21	6,898	26,79	10,05	4,896	3,775	3,499
9	19,02	6,751	26,79	10,06	4,887	3,715	3,527
10	19,12	6,787	26,76	10,04	4,887	3,738	3,432
$\mathbb{E}(T)$	19,096	6,8678	26,762	10,05	4,8917	3,7961	3,4568
RAM	0.84	1800	0.84	1800			

Definition 2. The acceleration $A_c(N)$ given by the parallel algorithm to the c - stream computing system, in comparison with the most efficient sequential algorithm,

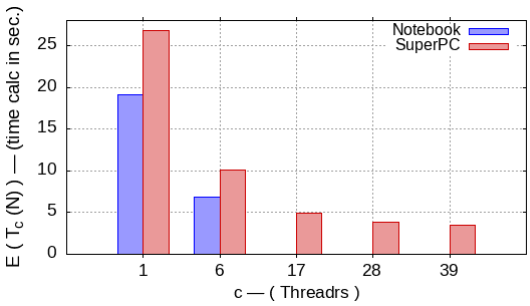


Fig. 2. Average calculation time in seconds, with different c – number of threads

is determined by the formula:

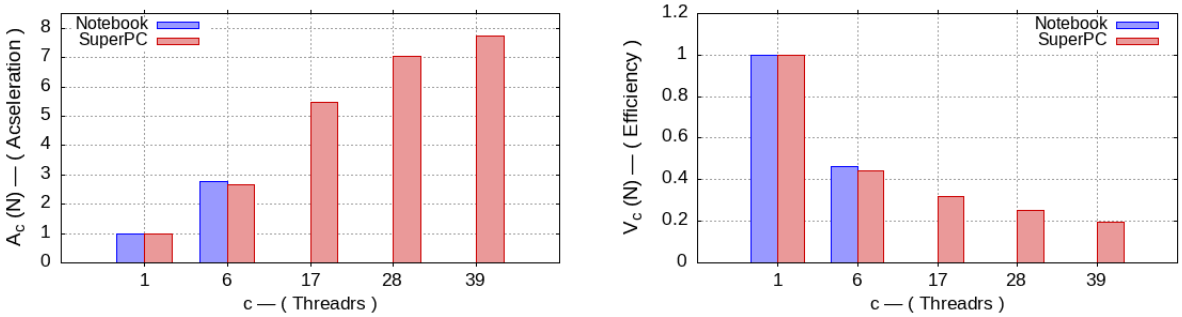
$$A_c(N) = \frac{\mathbb{E}(T_1(N))}{\mathbb{E}(T_c(N))}, \tag{8}$$

where, $\max(A_c(N)) = c$ is the theoretical value possible in the absence of unavoidable time delays in the execution of the parallel algorithm. Those. when there are no additional actions for interaction between threads, all calculations are evenly distributed among the threads and no action is required to combine the results of calculations.

Definition 3. The efficiency $V_c(N)$ of using a given number of threads by a parallel algorithm is determined by the formula:

$$V_c(N) = \frac{\mathbb{E}(T_1(N))}{c \cdot \mathbb{E}(T_c(N))} = \frac{A_c(N)}{c}, \tag{9}$$

where, for $\max A_c(N) = c$ we get that $\max(E_c(N)) = 1$.



а) б)

Fig. 3. Acceleration (8) and efficiency (9) calculated based on data from Table 1

Definition 4. The cost of $C_c(N)$ is the product of the number of threads used and $T_c(N)$ of the parallel algorithm solution time:

$$C_c(N) = \mathbb{E}(T_c(N)) \cdot c. \tag{10}$$

Definition 5. The cost-optimal index of the algorithm is characterized by the cost proportional to complexity of the best sequential algorithm [40], and in this case:

$$CO_c(N) = \frac{C_c(N)}{\mathbb{E}(T_1(N))}. \tag{11}$$

а) б)

Fig. 4. Cost (10) and cost Optimum Index (11) of algorithms

Remark 10. All figures 1–4 are built using scripts for `gnuplot` – is a freeware program, though not part of GNU under the GNU GPL.

From the results of the analysis of the effectiveness of the proposed EFDS algorithms based on the data (Figure 2 and Table 1) on the computation time during the numerical experiment, it can be seen that there is a significant acceleration of calculations (Figure 3 a). However, with an increase in the number of threads for parallel processing, their efficiency significantly decreases (Figure 3), and when running calculations on more than 15-20 threads, this does not give a performance increase. This also entails a deterioration in cost indexes (Figure 4), which characterize the ratio of the spent computing resources to the resulting acceleration.

Conclusions

The efficiency of the proposed parallel numerical algorithm was analyzed in comparison with its sequential analogue, which shows a significant 3-5-fold increase in performance for a non-local explicit finite-difference scheme. It can be concluded that it is more than possible to use powerful personal computers and laptops for efficient calculations of direct problems in fractional dynamics models based on nonlinear equations, in particular with the Gerasimov-Caputo fractional derivative VO. The use of super computers for solving direct tasks is also justified, but with some limitations.

References

[1] Acioli P. S., Xavier F. A., Moreira D. M. Mathematical Model Using Fractional Derivatives Applied to the Dispersion of Pollutants in the Planetary Boundary Layer, Boundary-Layer Meteorology, 2019, vol. 170, no. 2, pp. 285–304. DOI: 10.1007/s10546-018-0403-1.

[2] Aslam M., Farman M., Ahmad H., Gia T. N., Ahmad A., Askar S. Fractal fractional derivative on chemistry kinetics hires problem, AIMS Mathematics, 2021, vol. 7, no. 1, pp. 1155–1184. DOI: 10.3934/math.2022068.

[3] Jamil B., Anwar M. S., Rasheed A., Irfan M. MHD Maxwell flow modeled by fractional derivatives with chemical reaction and thermal radiation, Chinese Journal of Physics, 2020, vol. 67, pp. 512–533. DOI: 10.1016/j.cjph.2020.08.012.

[4] Fellah M., Fellah Z. E. A., Mitri F., Ogam E., Depollier C. Transient ultrasound propagation in porous media using Biot theory and fractional calculus: Application to human cancellous

bone, The Journal of the Acoustical Society of America, 2013, vol. 133, no. 4, pp. 1867–1881. DOI: 10.1121/1.4792721.

[5] Chen W. An Intuitive Study of Fractional Derivative Modeling and Fractional Quantum in Soft Matter, Journal of Vibration and Control, 2008, vol. 14, no. 9–10, pp. 1651–1657. DOI: 10.1177/1077546307087398.

[6] Garrappa R. Numerical Solution of Fractional Differential Equations: A Survey and a Software Tutorial, Mathematics, 2018, vol. 6, no. 2:16, pp. 1–23. DOI: 10.3390/math6020016.

[7] Bohaienko V. Parallel algorithms for modelling two-dimensional non-equilibrium salt transfer processes on the base of fractional derivative model, Fractional Calculus and Applied Analysis, 2018, vol. 21, no. 3, pp. 654–671. DOI: 10.1515/fca-2018-0035.

[8] Bogaenko V. A., Bulavatskiy V. M., Kryvonos I. G. On Mathematical modeling of Fractional-Differential Dynamics of Flushing Process for Saline Soils with Parallel Algorithms Usage, Journal of Automation and Information Sciences, 2016, vol. 48, no. 10, pp. 1–12. DOI: 10.1615/JAutomatInfScien.v48.i10.10.

[9] Gerasimov A. N. Generalization of linear deformation laws and their application to internal friction problems, Applied Mathematics and Mechanics, 1948, vol. 12, pp. 529–539.

[10] Caputo M. Linear models of dissipation whose Q is almost frequency independent – II, Geophysical Journal International, 1946, vol. 13, no. 5, pp. 529–539. DOI: 10.1111/j.1365-246X.1967.tb02303.x3.

[11] Sanders J., Kandrot E. CUDA by Example: An Introduction to General-Purpose GPU Programming. London, Addison-Wesley Professional, 2010, 311 pp.

[12] Bogaenko V. O. Parallel finite-difference algorithms for three-dimensional space-fractional diffusion equation with phi–Caputo derivatives, Computational and Applied Mathematics, 2020, vol. 39, no. 163, pp. 1–20. DOI: 10.1007/s40314-020-01191-x.

[13] Machado J. T., Kiryakova V., Mainardi F. Recent history of fractional calculus, Communications in nonlinear science and numerical simulation, 2011, vol. 16, no. 3, pp. 1140–1153. DOI: 10.1016/j.cnsns.2010.05.027.

[14] Patnaik S., Hollkamp J. P., Semperlotti F. Applications of variable-order fractional operators: a review, Proceedings of the Royal Society A, 2020, vol. 476, no. 2234, pp. 20190498. DOI: 10.1098/rspa.2019.0498.

[15] Ortigueira M. D., Valerio D., Machado J. T. Variable order fractional systems, Communications in Nonlinear Science and Numerical Simulation, 2019, vol. 71, pp. 231–243. DOI: 10.1016/j.cnsns.2018.12.003.

[16] Tverdyi D. A. The Cauchy problem for the Riccati equation with variable power memory and non-constant coeffcients, Bulletin KRASEC. Physical and Mathematical Sciences, 2018, vol. 23, no. 3, pp. 148–157. DOI: 10.18454/2079-6641-2018-23-3-148-157.

[17] Borzunov S. V., Kurgalin S. D., Flegel A. V. Praktikum po parallel'nomu programmirovaniyu: uchebnoe posobie [Workshop on Parallel Programming: A Study Guide]. Saint Petersburg: BVH, 2017, 236 pp., isbn: 978-5-9909805-0-1 (In Russian).

[18] Kilbas A. A., Srivastava H. M., Trujillo J. J. Theory and Applications of Fractional Differential Equations. Amsterdam, Elsevier Science Limited, 2006, 523 pp., isbn: 9780444518323.

[19] Tverdyi D. A., Parovik R. I. Investigation of Finite-Difference Schemes for the Numerical Solution of a Fractional Nonlinear Equation, Fractal and Fractional, 2022, vol. 6(1), no. 23, pp. 1–27. DOI: 10.3390/fractalfract6010023.

[20] Tvyordyj D. A. Hereditary Riccati equation with fractional derivative of variable order, Journal of Mathematical Sciences, 2021, vol. 253, no. 4, pp. 564–572. DOI: 10.1007/s10958-021-05254-0.

[21] Tverdyi D. A., Parovik R. I. Application of the Fractional Riccati Equation for Mathematical Modeling of Dynamic Processes with Saturation and Memory Effect, Fractal and Fractional, 2022, vol. 6(3), no. 163, pp. 1–35. DOI: 10.3390/fractalfract6030163.

[22] Daintith J., Wright E. A Dictionary of Computing. Oxford, Oxford University Press, 2008, 583 pp., isbn: 9780191726576. DOI: 10.1093/acref/9780199234004.001.0001.

[23] Miller R., Boxer L. Algorithms Sequential and Parallel: A Unified Approach. 3rd edition. Boston, Cengage Learning, 2013, 417 pp., isbn: 978-1133366805.

[24] Rauber T., Runger G. Parallel Programming for Multicore and Cluster Systems. 2nd edition. New York, Springer, 2013, 516 pp., isbn: 978-3-642-37800-3.

[25] Al-hayanni M. A. N., Xia F., Rafiev A., Romanovsky A., Shafik R., Yakovlev A. Amdahl's Law in the Context of Heterogeneous Many-core Systems - ASurvey, IET Computers & Digital Techniques, 2020, vol. 14, no. 4, pp. 133–148. DOI: 10.1049/iet-cdt.2018.5220.

[26] Okrepilov V. V., Makarov V. L., Bakhtizin A. R., Kuzmina S. N. Application of Supercomputer Technologies for Simulation of Socio-Economic Systems, R-Economy, 2015, vol. 1, no. 2, pp. 340–350. DOI: 10.15826/recon.2015.2.016.

[27] Il'in V. P., Skopin I. N. About performance and intellectuality of supercomputer modeling, Programming and Computer Software, 2016, vol. 42, no. 1, pp. 5–16. DOI: 10.1134/S0361768816010047.

[28] Parovik R. I. On the numerical solution of equations fractal oscillator with variable order fractional of time, Bulletin KRASEC. Physical and Mathematical Sciences, 2014, vol. 8, no. 1, pp. 60–65. DOI: 10.18454/2079-6641-2014-8-1-60-65.

[29] Parovik R. I. Mathematical models of oscillators with memory, Oscillators-Recent Developments, 2019, pp. 3–21. DOI: 10.5772/intechopen.81858.

[30] Volterra V. Theory of functionals and of integral and integro-differential equations: [Unabridged republication of the first English translation]. New York, Dover publications, 1959, 226 pp.

[31] Uchaikin V. V. Fractional Derivatives for Physicists and Engineers. Vol. I. Background and Theory. Berlin, Springer, 2013, 373 pp., isbn: 978-3-642-33911-0. DOI: 10.1007/978-3-642-33911-0.

[32] Tverdyi D. A., Makarov E. O., Parovik R. I. Hereditary Mathematical Model of the Dynamics of Radon Accumulation in the Accumulation Chamber, Mathematics, 2023, vol. 11, no. 4:850, pp. 1–20. DOI: 10.3390/math11040850.

[33] Jeng S., Kilicman A. Fractional Riccati Equation and Its Applications to Rough Heston Model Using Numerical Methods, Symmetry, 2020, vol. 12. DOI: 10.3390/sym12060959.

[34] Sun H., et al. Finite difference schemes for variable-order time fractional diffusion equation, International Journal of Bifurcation and Chaos, 2012, vol. 22, no. 04, pp.1250085. DOI: 10.1142/S021812741250085X.

[35] Parovik R. I. On a finite-difference scheme for an hereditary oscillatory equation, Journal of Mathematical Sciences, 2021, vol. 253, no. 4, pp. 547–557. DOI: 10.1007/s10958-021-05252-2.

[36] Kalitkin N. N. Chislennye metody. 2-e izd. [Numerical methods. 2nd ed.]. Saint Petersburg: BVH, 2011, 592 pp., isbn: 978-5-9775-0500-0 (In Russian).

[37] Brent R. P. The parallel evaluation of general arithmetic expressions, Journal of the Association for Computing Machinery, 1974, vol. 21, no. 2, pp. 201–206. DOI: 10.1145/321812.321815.

[38] Corman T. H., Leiserson C. E., Rivet R. L., Stein C. Introduction to Algorithms, 3rd Edition. Cambridge, The MIT Press, 2009, 1292 pp., isbn: 978-0262033848.

[39] Shao J. Mathematical Statistics. 2-ed. New York, Springer, 2003, 592 pp., isbn: 978-0-387-95382-3.

[40] Gergel V. P., Strongin R. G. Vysokoproizvoditel'nye vychisleniya dlya mnogoyadernyh mnogoprocessornyh sistem. Uchebnoe posobie [High performance computing for multi-core multiprocessor systems. study guide]. Moscow: MGU publishing, 2010, 544 pp.,(In Russian).

Information about authors

Tverdyi Dmitrii Alexsandrovich – Ph. D. (Phys. & Math.), Researcher of the International Integrative Research Laboratory of Extreme Phenomena of Kamchatka, Vitus Bering Kamchatka State University, Petropavlovsk-Kamchatsky, Russia, ORCID 0000-0001-6983-5258.

Parovik Roman Ivanovich – D. Sci. (Phys. & Math.), Associate Professor, Head of the International Integrative Research Laboratory of Extreme Phenomena of Kamchatka, Vitus Bering Kamchatka State University, Petropavlovsk-Kamchatsky, Russia, ORCID 0000-0002-1576-1860.

Hayotov Abdullo Rahmonovich – D. Sci. (Phys. & Math.), Associate Professor, Laboratory manager Scientific laboratory of computational mathematics Institute of Mathematics named after V.I. Romanovsky AS RUz, 683023, Tashkent, Uzbekistan, 0000-0002-2756-9542.

Boltayev Aziz Qo'ziyevich – Ph. D. (Phys. & Math.), Senior Researcher Scientific laboratory of computational mathematics Institute of Mathematics named after V.I. Romanovsky AS RUz, 683023, Tashkent, Uzbekistan, 0000-0002-8329-4440.

References

1. Acioli P. S., Xavier F. A., Moreira D. M. Mathematical Model Using Fractional Derivatives Applied to the Dispersion of Pollutants in the Planetary Boundary Layer, *Boundary-Layer Meteorology*, 2019. Т. 170, №2, С. 285–304 DOI: 10.1007/s10546-018-0403-1.
2. Aslam M., Farman M., Ahmad H., Gia T. N., Ahmad A., Askar S. Fractal fractional derivative on chemistry kinetics hires problem, *AIMS Mathematics*, 2021. Т. 7, №1, С. 1155–1184 DOI: 10.3934/math.2022068.
3. Jamil B., Anwar M. S., Rasheed A., Irfan M. MHD Maxwell flow modeled by fractional derivatives with chemical reaction and thermal radiation, *Chinese Journal of Physics*, 2020. Т. 67, С. 512–533 DOI: 10.1016/j.cjph.2020.08.012.
4. Fellah M., Fellah Z. E. A., Mitri F., Ogam E., Depollier C. Transient ultrasound propagation in porous media using Biot theory and fractional calculus: Application to human cancellous bone, *The Journal of the Acoustical Society of America*, 2013. Т. 133, №4, С. 1867–1881 DOI: 10.1121/1.47927213.
5. Chen W. An Intuitive Study of Fractional Derivative Modeling and Fractional Quantum in Soft Matter, *Journal of Vibration and Control*, 2008. Т. 14, №9–10, С. 1651–1657 DOI: 10.1177/1077546307087398.
6. Garrappa R. Numerical Solution of Fractional Differential Equations: A Survey and a Software Tutorial, *Mathematics*, 2018. Т. 6, №2:16, С. 1–23 DOI: 10.3390/math6020016.
7. Bohaienko V. Parallel algorithms for modelling two-dimensional non-equilibrium salt transfer processes on the base of fractional derivative model, *Fractional Calculus and Applied Analysis*, 2018. Т. 21, №3, С. 654–671 DOI: 10.1515/fca-2018-0035.
8. Bogaenko V. A., Bulavatskiy V. M., Kryvonos I. G. On Mathematical modeling of Fractional-Differential Dynamics of Flushing Process for Saline Soils with Parallel Algorithms Usage, *Journal of Automation and Information Sciences*, 2016. Т. 48, №10, С. 1–12 DOI: 10.1615/JAutomatInfScien.v48.i10.10.
9. Gerasimov A. N. Generalization of linear deformation laws and their application to internal friction problems, *Applied Mathematics and Mechanics*, 1948. Т. 12, С. 529–539.
10. Caputo M. Linear models of dissipation whose Q is almost frequency independent – II, *Geophysical Journal International*, 1967. Т. 13, №5, С. 529–539 DOI: 10.1111/j.1365-246X.1967.tb02303.x.
11. Sanders J., Kandrot E. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. London: Addison-Wesley Professional, 2010. 311 с.
12. Bogaenko V. O. Parallel finite-difference algorithms for three-dimensional space-fractional diffusion equation with phi–Caputo derivatives, *Computational and Applied Mathematics*, 2020. Т. 39, №163, С. 1–20 DOI: 10.1007/s40314-020-01191-x.
13. Machado J. T., Kiryakova V., Mainardi F. Recent history of fractional calculus, *Communications in nonlinear science and numerical simulation*, 2011. Т. 16, №3, С. 1140–1153 DOI: 10.1016/j.cnsns.2010.05.027.
14. Patnaik S., Hollkamp J. P., Semperlotti F. Applications of variable-order fractional operators: a review, *Proceedings of the Royal Society A*, 2020. Т. 476, №2234, С. 20190498 DOI: 10.1098/rspa.2019.0498.
15. Ortigueira M. D., Valerio D., Machado J. T. Variable order fractional systems, *Communications in Nonlinear Science and Numerical Simulation*, 2019. Т. 71, С. 231–243 DOI: 10.1016/j.cnsns.2018.12.003.
16. Твёрдый Д. А. Задача Коши для уравнения Риккати с непостоянными коэффициентами и учетом переменной степенной памяти, *Вестник КРАУНЦ. Физико-математические науки*, 2018. Т. 23, №3, С. 148–157 DOI: 10.18454/2079-6641-2018-23-3-148-157.
17. Борзунов С. В., Кургалин С. Д., Флегель А. В. *Практикум по параллельному программированию: учебное пособие*. Санкт-Петербург: БХВ, 2017. 236 с. ISBN 978-5-9909805-0-1.
18. Kilbas A. A., Srivastava H. M., Trujillo J. J. *Theory and Applications of Fractional Differential Equations*. Amsterdam: Elsevier Science Limited, 2006. 523 с. ISBN 9780444518323.
19. Tverdyi D. A., Parovik R. I. Investigation of Finite-Difference Schemes for the Numerical Solution of a Fractional Nonlinear Equation, *Fractal and Fractional*, 2022. Т. 6(1), №23, С. 1–27 DOI: 10.3390/fractalfract6010023.
20. Tvyordyj D. A. Hereditary Riccati equation with fractional derivative of variable order, *Journal of Mathematical Sciences*, 2021. Т. 253, №4, С. 564–572 DOI: 10.1007/s10958-021-05254-0.

21. Tverdyi D. A., Parovik R. I. Application of the Fractional Riccati Equation for Mathematical Modeling of Dynamic Processes with Saturation and Memory Effect, *Fractal and Fractional*, 2022. T. 6(3), № 163, С. 1–35 DOI: 10.3390/fractalfract6030163.
22. Daintith J., Wright E. *A Dictionary of Computing*. Oxford: Oxford University Press, 2008. 583 с. ISBN 9780191726576 DOI: 10.1093/acref/9780199234004.001.0001.
23. Miller R., Boxer L. *Algorithms Sequential and Parallel: A Unified Approach. 3rd edition*. Boston: Cengage Learning, 2013. 417 с. ISBN 978-1133366805.
24. Rauber T., Runger G. *Parallel Programming for Multicore and Cluster Systems. 2nd edition*. New York: Springer, 2013. 516 с. ISBN 978-3-642-37800-3.
25. Al-hayanni M. A. N., Xia F., Rafiev A., Romanovsky A., Shafik R., Yakovlev A. Amdahl's Law in the Context of Heterogeneous Many-core Systems - ASurvey, *IET Computers & Digital Techniques*, 2020. Т. 14, № 4, С. 133–148 DOI: 10.1049/iet-cdt.2018.5220.
26. Okrepilov V. V., Makarov V. L., Bakhtizin A. R., Kuzmina S. N. Application of Supercomputer Technologies for Simulation of Socio-Economic Systems, *R-Economy*, 2015. Т. 1, № 2, С. 340–350 DOI: 10.15826/recon.2015.2.016.
27. Il'in V. P., Skopin I. N. About performance and intellectuality of supercomputer modeling, *Programming and Computer Software*, 2016. Т. 42, № 1, С. 5–16 DOI: 10.1134/S0361768816010047.
28. Паровик Р. И. О численном решении уравнения фрактального осциллятора с производной дробного переменного порядка от времени, *Вестник КРАУНЦ. Физико-математические науки*, 2014. Т. 8, № 1, С. 60–65 DOI: 10.18454/2079-6641-2014-8-1-60-65.
29. Parovik R. I. Mathematical models of oscillators with memory, *Oscillators-Recent Developments*, 2019, С. 3–21 DOI: 10.5772/intechopen.81858.
30. Volterra V. *Theory of functionals and of integral and integro-differential equations*. New York: Dover publications, 1959. 226 с.
31. Uchaikin V. V. *Fractional Derivatives for Physicists and Engineers. Vol. I. Background and Theory*. Berlin: Springer, 2013. 373 с. ISBN 978-3-642-33911-0 DOI: 10.1007/978-3-642-33911-0.
32. Tverdyi D. A., Makarov E. O., Parovik R. I. Hereditary Mathematical Model of the Dynamics of Radon Accumulation in the Accumulation Chamber, *Mathematics*, 2023. Т. 11, № 4:850, С. 1–20 DOI: 10.3390/math11040850.
33. Jeng S., Kilicman A. Fractional Riccati Equation and Its Applications to Rough Heston Model Using Numerical Methods, *Symmetry*, 2020. Т. 12 DOI: 10.3390/sym12060959.
34. Sun H., et al. Finite difference schemes for variable-order time fractional diffusion equation, *International Journal of Bifurcation and Chaos*, 2012. Т. 22, № 04, С. 1250085 DOI: 10.1142/S021812741250085X.
35. Parovik R. I. On a finite-difference scheme for an hereditary oscillatory equation, *Journal of Mathematical Sciences*, 2021. Т. 253, № 4, С. 547–557 DOI: 10.1007/s10958-021-05252-2.
36. Калиткин Н. Н. *Численные методы. 2-е изд.*. Санкт–Петербург: БХВ, 2011. 592 с. ISBN 978-5-9775-0500-0.
37. Brent R. P. The parallel evaluation of general arithmetic expressions, *Journal of the Association for Computing Machinery*, 1974. Т. 21, № 2, С. 201–206 DOI: 10.1145/321812.321815.
38. Corman T. H., Leiserson C. E., Rivet R. L., Stein C. *Introduction to Algorithms, 3rd Edition*. Cambridge: The MIT Press, 2009. 1292 с. ISBN 978-0262033848.
39. Shao J. *Mathematical Statistics. 2-ed*. New York: Springer, 2003. 592 с. ISBN 978-0-387-95382-3.
40. Гергель В. П. *Высокопроизводительные вычисления для многоядерных многопроцессорных систем. Учебное пособие*. Москва: Издательство МГУ, 2010. 544 с.

Информация об авторах

Твёрдый Дмитрий Александрович – кандидат физико-математических наук, научный сотрудник международной интегративной научно-исследовательской лаборатории экстремальных явлений Камчатки Камчатского государственного университета им. Витуса Беринга, Петропавловск-Камчатский, Россия, ORCID 0000-0001-6983-5258.

Паровик Роман Иванович – доктор физико-математических наук, доцент, заведующий международной интегративной научно-исследовательской лаборатории экстремальных явлений Камчатки Камчатского государственного университета им. Витуса Беринга, Петропавловск-Камчатский, Россия, ORCID 0000-0002-1576-1860.

Хаётов Абдулло Рахмонович – доктор физико-математических наук, профессор, заведующий научной лабораторией вычислительной математики Института математики имени В.И. Романовского АН РУз, г. Ташкент, Республика Узбекистан, 0000-0002-2756-9542.

Болтаев Азиз Козиевич – кандидат физико-математических наук, старший научный сотрудник научной лаборатории вычислительной математики Института математики имени В.И. Романовского АН РУз, г. Ташкент, Республика Узбекистан, 0000-0002-8329-4440.